PB86-155785

Plant Exposure Laboratory and Chambers. Volume 2 Appendices 1 through 6

(U.S.) Corvallis Environmental Research Lab., OR

Jan 86

U.S. Department of Commerce
National Technical Information Service
NTIS

PB86-155785

EPA/600/3-86/007b
January 1986

PLANT EXPOSURE LABORATORY AND CHAMBERS

Volume II. Appendices 1 through 6

CORVALLIS ENVIRONMENTAL RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CORVALLIS, OREGON 97333

REPRODUCED BY
NATIONAL TECHNICAL
INFORMATION SERVICE
U.S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

TECHNICAL REPORT DATA
(Please read Instructions on the reverse before completing)

1. REPORT NO. EPA/600/3-86/007b	**2.**	**3. RECIPIENT'S ACCESSION NO.** PB86 155785/AS
4. TITLE AND SUBTITLE PLANT EXPOSURE LABORATORY AND CHAMBERS: Volume II. Appendices 1 through 6		**5. REPORT DATE** January 1986 **6. PERFORMING ORGANIZATION CODE**
7. AUTHOR(S) C.McFarlane, T. Pfleeger, M. McDowell, J. Wilson and R. Shimabuku		**8. PERFORMING ORGANIZATION REPORT NO.**
9. PERFORMING ORGANIZATION NAME AND ADDRESS ERL-Corvallis		**10. PROGRAM ELEMENT NO.** **11. CONTRACT/GRANT NO.**
12. SPONSORING AGENCY NAME AND ADDRESS Environmental Research Laboratory Office of Research and Development U.S.Environmental Protection Agency Corvallis, OR 97333		**13. TYPE OF REPORT AND PERIOD COVERED** In-house Report **14. SPONSORING AGENCY CODE** EPA 600/02
15. SUPPLEMENTARY NOTES 1985. Volume I of this report is the text.		

16. ABSTRACT

This paper describes the laboratory and support system, the exposure chambers, the computer system, and the hydroponic nursery built to accomplish research on uptake, translocation, and metabolism of toxic organic chemicals by plants. Understanding these processes and the physiological driving mechanisms is essential to predict food chain contamination and environmental damage from various agricultural and industrial pollutants. Since the chemicals of interest are toxic and studies generally include 14C as a label for minitoring chemical kinetics, containment was an important criterion. Containment and safety considerations are included.

17. KEY WORDS AND DOCUMENT ANALYSIS

a. DESCRIPTORS	b. IDENTIFIERS/OPEN ENDED TERMS	c. COSATI Field/Group

18. DISTRIBUTION STATEMENT Release to Public	**19. SECURITY CLASS** *(This Report)* Unclassified **20. SECURITY CLASS** *(This page)* Unclassified	**21. NO. OF PAGES** 285 **22. PRICE**

APPENDIX 1

CONTROL PROGRAM: USER'S GUIDE

Computer Programs used to manage equipment and data in the Plant Exposure Laboratory.

by

Mike McDowell

Toxic and Hazardous Materials Branch
Corvallis Environmental Research Laboratory
200 S. W. 35th St.
Corvallis, Oregon 97333

CORVALLIS ENVIRONMENTAL RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CORVALLIS, OREGON 97333

ACKNOWLEDGEMENTS

I would like to thank Brian Martin for the section on the Select/Merge program, Robert Clark for the section on the telephone alert procedure and Craig Mc Farlane for his help with writing style and his editing expertise.

DISCLAIMER

The information in this document has been funded wholly or in part by the U.S. Environmental Protection Agency. It has been subjected to the Agency's peer and administrative reviews and has been approved for publication as an EPA document. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

CONTENTS

Page

Introduction 1-1

Conventions 1-1

Output 1-1

Input 1-2

The main menu 1-4

Define parameters 1-8

Parameters description 1-11

Measurement series 1-17

Air measurement 1-18

Data collection 1-19

Statistics collection and display 1-21

Examine 1-24

Test devices 1-27

Telephone operation 1-28

Select/Merge 1-29

FIGURES

Page

Main menu 1-4

Set clock menu 1-6

Define parameters menu 1-8

Edit positions menu 1-9

One position/All channels summary statistics display 1-23

One channel/All positions summary statistics display 1-24

Test devices menu 1-27

Select change options menu 1-30

Select change other options menu 1-31

CONTROL USER'S GUIDE

INTRODUCTION

The CONTROL system of programs is designed to measure and regulate the plant exposure chambers and laboratory. The computer monitors up to 18 exposure chambers, each with 16 input channels of information, and also adjusts various conditions in each chamber as needed. Descriptive statistics are continually updated so that the state of the system is evident at a glance and a periodic summary is stored. For more thorough analysis, a graphics routine is available, and CONTROL interfaces with a complete statistics, graphics, and database program, Statpro (version 1.1, Wadsworth Electronic Publishing Co.).

No programming experience is needed to use CONTROL. However, it is helpful to be familiar with the program's operating system (Apple Pascal 1.2) when using CONTROL. In particular, chapters 1, 2, 3, and pages 184-186 (formatting diskettes) in the Apple Pascal Operating System Reference manual should be read. Chapter 4 is also recommended.

CONVENTIONS

Throughout this guide, certain conventions and abbreviations are used.

BRACKETS surrounding a letter or key designation refer to that key on the keyboard. For example, {-->} refers to the right arrow key, {space} refers to the space bar, and {A} refers to the 'A' key.

CONTROL CHARACTERS are sometimes used. To type a control character, press and hold the {control} key while simultaneously pressing the necessary key. This operation is designated with ctrl, a dash, and the key to be pressed. For example, ctrl-W means press and hold the {control} key while pressing the {W} key.

PROMPTS from the program are designated with quotes. For example, a program prompt will appear as follows: "Enter a value between 3 and 6".

OUTPUT

Output is in two forms; data stored on the datafile and information displayed on the screen. The datafiles are collections of sequential records, one record per chamber. Each record holds 36 values which include identifying information and data. There are up to 30,000 records in each datafile.

Summary statistics are stored in a separate file. A detailed description is on pages 21-24.

Different screen outputs can be selected from the CONTROL main menu. Options are: summary statistics display, graphical data displays, and various CONTROL parameter menus. The menus allow implementation/de-implementation of program options and access to different portions of the program by pressing one key from the correct menu. Different menus are linked together so that some program options are accessible by stepping through a series of menus. The program also displays menus which allow the user to edit program parameters. Each menu will be described individually.

INPUT

Input for CONTROL comes in two forms; data from the experimental chambers, and input from the keyboard.

All keyboard inputs are processed by a special subroutine which checks for the correct type of instruction. For example, if the program expects a number, it will not accept letters. This subroutine also accepts special commands which allow movement around the screen in order to edit parameters.

SELECTING OPTIONS FROM A MENU. Options on a menu are preceded by a letter. Choose the option by pressing that letter.

MOVING AROUND THE SCREEN. There are four ways to move around the screen. The left and right arrows on all Apple //'s allow left and right movement. On the Apple //+, use ctrl-o to move up and ctrl-l to move down. On the Apple //e or //c, use the up and down arrow keys to move up and down.

ENTERING A NEW PARAMETER VALUE. When you have reached the parameter which you wish to edit, just type in the new value. There are three ways to make the computer recognize that you have finished the entry:

{return} is the normal way to finish an entry. By pressing {return} , the computer checks and accepts a legal entry. If it is not legal, the computer beeps indicating an invalid command.

{-->} also finishes an entry. The computer checks and accepts the entry as with {return} and in addition, moves one item to the right. Pressing {-->} to finish an entry works the same as pressing {return} and then pressing {-->} .

{+} is a special command which terminates an entry. In some areas of the program, a {+} moves from a screen to a subsequent screen. Pressing {+} will check and accept an entry just as with {return} but will then move to the subsequent screen.

There are three more special commands used during editing:

{esc} is a special terminator. It REJECTS any entry, replaces it with the original entry, and leaves the edit screen, usually returning to a menu.

{space} ends NUMBER entries the same as {return}.

{<--} is used to back up to correct mistakes made when typing in a new entry. However, from the left-most position of an entry, the cursor is moved left one position by pressing {<--} .

SUMMARY OF DATA ENTRY.

{-->}: Move one item to the right. Finish a data entry and move one item to the right.

{<--}: Move one item to the left. Backspace to correct typing errors.

{space}: Finish a number data entry. Insert spaces into text entries.

{esc}: Finish all data entry; return to menu.

{up}: Move one item up. Finish a data entry and move one item up.

{+}: Move to the next screen of information when editing. Finish a data entry and move to the next screen of information.

{down}: Move one item down. Finish a data entry and move one item down.

{return}: Finish a data entry.

THE MAIN MENU

```
Time: [12:06:24]  Date: [17 Dec 84]  Memory: [7300]  Record: [1245]
Air system [ Purging]  Stats [-]  Data [+]  Regular [+]  Air [+]  Phone [-]

                          -- Main menu --

                             Options:
                             -------
  A) regular INTERRUPTS on/off        I) Define Parameters
  B) AIR INTERRUPTS on/off            J) Define Parameters (abbrev.)
  C) PHONE answering on/off           K) Set Clock
  D) Data collection on/off           L) Initialize new data file
  E) Statistics collection on/off     M) Initialize new stats file
  F) Examine                          N) Force new statistics period
  G) Measure                          O) Select/Merge data file to diskette
  H) Test Devices                     P) Stat display position/channel edit
                                      S) Volume number edit for workfiles

<--/--> Select Screen;   R)eturn to Pascal;   Error:  [ 0 ]
```

DESCRIPTION. By pressing one of the keys indicated by a right parenthesis, the user selects that option (i.e. pressing {A} selects: A) regular INTERRUPTS on/off).

The menu also displays current program information. At the top of the screen (line 1) the time, date, the amount of empty memory, and the current record number in the workfile are displayed. On line 2, the status of several program functions is indicated. They correspond to the on/off switches operated by functions A) through E) on the main menu. The last line displays the current error condition (usually 0).

<--/--> Select Screen option. CONTROL has three main menu displays; these displays do not interfere with the main menu options which can be selected at any time from any of these menus. Right and left arrows allow selection of these alternate screens. Generally, the program displays the main menu as the default display. However, if statistics are being collected, the two different statistics display screens can be selected as the default display by using the arrow keys. Pressing {-->} from the main menu selects the first statistics screen, pressing {-->} a second time selects the second statistics screen. Pressing {-->} a third time returns you to the main menu. The screens can be selected in the opposite direction by pressing {<--}. The contents of the statistics display screens is described in more detail in the statistics collection section, pages 21-24.

MENU OPTIONS. Note on 'toggling'. Some of the options described below use a toggle option to turn a program feature on or off. If the feature is on and you wish to turn it off, respond by pressing {Y} to the "Toggle? [y/n]" prompt. Respond the same way if the feature is off and you wish to turn it on. Pressing {N} to the prompt will not change the status of the feature.

A. regular INTERRUPTS on/off. When on, the program interrupts normal functioning after a designated period (sample rate parameter, miscellaneous record; see the parameters description section, page 14) to take chamber measurements and perform whatever environmental control is necessary (see page 17). When off, the measurements are not performed. The status of this feature is indicated in line two of the main screen as "Regular [+/-]".

B. AIR INTERRUPTS on/off. When on, this function causes the program to perform the air cycling loop (CO_2 measurement and control, see the section on pages 18-19) for each on-line cart (air sampling parameter, position record; see the parameters description section, page 11). When off, air measurement and control does not occur on any of the carts. The status is indicated in line two as "Air [+/-]".

C. PHONE answering on/off. The program is designed to allow computer phone communication. By telephoning the lab and using the proper communications program, the user can access CONTROL as if actually in the laboratory. When this option is turned off, telephone communication is prohibited. The status of this feature is indicated in line two as "Phone [+/-]".

D. Data collection on/off. With this feature turned off, the program will continue running (measurements will be made and the chambers which are online will be environmentally controlled) but no data will be stored on the workfile. When on, the program will function and collect data normally. When data collection is turned off, the workfile will be updated and closed. When it is turned on, the existing workfile will be opened. If there is no existing workfile, it will appear as if nothing happened, no workfile will be opened. If this is the case, you must initialize a new workfile, option L). The status of this feature is indicated in line two as "Data [+/-]".

E. Statistics collection on/off. Since statistical summing does take some time, it might be beneficial at certain times to keep this feature turned off. Otherwise statistics will be collected for every measured channel and every online chamber in a special statistics workfile. See the statistics collection section, pages 21-24. The status of this is indicated in line two as "Stats [+/-]".

F. Examine (DISPLAY DATA). This feature is a complete program in itself. When {F} is pressed, CONTROL is suspended and the Examine menu is displayed. Examine allows graphic presentation of the data that has been collected on the workfile. Any part of the workfile can be displayed, with up to six channels of information on the screen at one time. When finished with Examine, CONTROL will pick up from where it left off. See the Examine section, pages 24-27, for a complete description of this option.

G. Measure. Although CONTROL is configured to run automatically and collect measurements at prescribed intervals, this option gives the user the opportunity to take a measurement at any time by pressing {G}. The measurement screen will appear and all chambers online at that time will be measured; the data collected will be saved on the workfile just as if a regular interrupt and measurement had occurred. While the measurements from one of the chambers are being displayed, {esc} can be pressed to abort the measurement process, or {W} can be pressed to hold the current display on the screen until {space} is pressed. See the measurement section, pages 17-18, for a complete description of the measurement process.

H. Test devices. This routine allows the user to completely test the hardware devices which are connected to the computer. Press {H} to get the test devices menu. Details are described in Test devices section, pages 27-28.

I. and J. Define parameters. Both of these items allow editing and review of all the parameters which specify how CONTROL will operate. I) is a complete program, while J) is a shortened version which is a part of CONTROL. The advantage of option I) is that it has superior editing power, but works somewhat slowly, while option J) is quicker at the cost of some of the editing functions. See the Define parameters section, pages 8-11, for details.

K. Set Clock. This option is usually only necessary to change the clock for daylight savings time. Press {K} to go to the set clock menu:

```
              -- Set Clock --

     Year              [85]  [0..99]
     Month             [ 2]  [1..12]
     Day               [ 5]  [1..31]
     Hours             [13]  [0..23]
     Minutes           [10]  [0..59]
     Day of Week       [ 3]  [Sun=1..Sat=7]
```

This is the current time according to the clock inside the computer. To change any these values, just move to the appropriate location on the screen and enter the new value. Press {esc} to update the clock to the time and date represented on the screen and return to the main menu. Please note that the clock is stopped while this routine is being used.

L. Initialize new data file. The workfile should be initialized only when the current workfile is full or a new experiment is being started. In either case, the data that was stored on the old workfile should be archived before it is removed. Pressing {L} brings up the prompt "Remove old data file? [Y/N] ". Press {Y} only to remove the old data file completely and start a new one in its place.

M. Initialize new stats file. This option operates just as option L) operates, except this applies to the statistics workfile. Be sure that the data has been saved before removing the old statistics file. Pressing {M} brings up the prompt "Remove old stats file? [Y/N] ". Press {Y} only if the old statistics file is to be removed completely and a new one started in its place.

N. Force new statistics period. At times you may want to end the current statistics collection period, save the data, and start a new statistical period. Press {N} to do this.

O. Select/Merge data file to diskette. This program allows selection of specific records from the data file to create new workfiles for archiving data, or for efficient data plotting and analysis. Press {O} to suspend CONTROL and start the Select program. See the Select/Merge section, pages 29-31, for more description on this program.

P. Stat display position/channel edit. This option displays this line at the bottom of the main menu:

```
"[2] channel [7] position; <--/--> move; <esc>".
```

The channel or position to display for the summary statistics screen can be entered here. Move to the item to edit, enter the new number, press {return} . When finished, press {esc} . The changes will be reflected the next time a summary statistics screen is displayed.

S. Volume number edit for workfiles. When {S} is pressed from the main menu, the following will be displayed at the bottom of the screen:

```
"[11] workfile [12] stat file; <--/--> move; <esc>".
```

The volume numbers (see description of volumes in the Apple Pascal Operating System Manual, pages 26-27) for the regular workfile and the statistics workfile can be changed using this option. Move to the number to edit, enter the new number, press {return}. Press {esc} when finished. This option will have to be used when regular disk drives are being used in place of the Corvus disk drive.

DEFINE PARAMETERS

There are two versions of the Define parameters routine, a long version which is a program in itself and a shortened version which is part of CONTROL. The longer version contains more options which ease parameter editing.

To edit a certain parameter, first move to that parameter using the movement commands as described in the Input section, pages 2-3. Then enter the new value. Sometimes, values are changed by pressing a certain key. Check all prompts which appear at the bottom of the screen before typing. Press {esc} to return to the previous menu.

DEFINE PARAMETERS PROGRAM (long version). This option allows editing and review of all the parameters which affect the function of CONTROL. Pressing {I} suspends CONTROL, loads the parameter file into memory, and displays the Define parameters menu.

```
            -- Define Parameters Menu --

        a) Errors
        b) Channels
        c) Miscellaneous
        d) Switches
        e) Positions
        f) Range of fatal errors
        g) Current program values
        h) Conversion equation parameters

               [ ] Enter choice
```

Please see the parameters description section, pages 11-17, for a description of the parameters, how they affect CONTROL, and a description of the different record types in the parameter file.

In this menu, the different options allow editing of the different records in the parameter file as follows:

a. Along with option f), this allows editing of the errors record. The parameters to edit are:

   Time of last error,
   Position of last error,
   Channel of last error,
   Telephone alert calling on/off, and
   Interval between phone calls.

b. This option allows editing of two parts of the channels record, the channel name and the channel scale. The maxmimum or minimum graphics scaling factor is edited in the Examine program (see the section on pages 24-27).

c. Edit the miscellaneous record. The following parameters can be edited:

Air sample rate,
Regular sample rate,
Transition period length,
Cart with light sensor,
Missing data flag,
CO_2 flow rate,
Relative period of statistics to display,
Position for statistics display,
Channel for statistics display, and
Number of standard gas measurements to skip.

d. Edit the switches record. Edit the name of each switch.

e. This option, along with option h), allows editing of the position records. Pressing {e} displays the following menu:

```
            -- Edit Positions --

a)  All positions, selected parameters
b)  One position, all parameters
c)  One position, conversion equations
d)  Load cart parameters
e)  Save cart parameters

            Enter choice [ ]
```

These options allow a variety of ways to edit the position parameters.

a. Pressing {a} displays a menu which is a list of the parameters which are contained in each position record. Select the parameter to edit by pressing the appropriate key. A screen will be displayed with that parameter value for every position. Move to the position to edit and edit the value according to the instructions which are at the bottom of the screen. Press {esc} to return to the Edit Positions menu.

b. To review and edit all the parameters on one cart only, press {b}. This displays the cart number, position number and all the other information which is held in the position record for that one position. Press {+} to display the next position record, or edit the position number itself to display the desired position record. Any value in the record can be edited by moving to the appropriate spot and following the directions which appear at the bottom of the display. Press {esc} to return to the Edit Positions menu.

c. One position, conversion equations. This option displays the conversion equations for each channel on each cart. Press {+} to display/edit the next channel in the series or enter a new channel or position number in the appropriate places on the screen. Note that there is a better routine under option h) on the Define parameters menu for editing these conversion equations. Press {esc} to return to the Edit Positions menu.

d. Load cart parameters. Pressing {d} allows a previously saved set of cart parameters (see option e, below) to be loaded into one of the positions in the parameter file. Enter the number of the cart which has been saved and the position number where the cart should be loaded. That cart's parameters will be loaded into the correct position in the parameter file. The Edit Positions menu will then be displayed.

e. Save cart parameters. Press {e} to save one of the position records onto the disk drive. Enter the position number of the cart to save. The program will load that position's parameters and save the record with the name being the number of the cart that was in that position. For example, if chamber 3 was placed in position 6, and position 6 was chosen as the record to save, it would be saved with the file name "CART3". To move cart 3 from position 6 to position 12, use option d) to load the parameters from cart 3 into position 12. The Edit Positions menu is automatically displayed when finished.

When finished editing positions, press {esc} to return to the Define Parameters menu.

f. Press {f} from the Define parameters menu to edit the Range of fatal errors parameters which are stored in the errors record. The channel names, and the current maximum and minimum values for the range will be displayed. Move to the parameter to change and enter the new value. Press {esc} to return to the Define parameters menu.

g. Press {g} to review and edit some of the current program values. These values are:

Last light measurement,
Current state of the air cycle,
Chamber in the air cycle,
Screen number,
Current period, and
Number of standard gas measurements skipped.

h. The complete set of conversion equations for one chamber can be reviewed and changed with this routine. The position and cart numbers, and the conversion equation parameters for each channel are displayed. Press {+} to display the next position's equations or edit the position number to display the set of equations for that position. Move to the parameter to edit, and enter the new value. When finished, press {esc} to return to the Define Parameters menu.

When done with the Define Parameters program, press {Q} to update the parameter file and return to the Control program.

DEFINE PARAMETERS ROUTINE (short version). The only differences between the two ways to edit parameters is that the short version does not have option h) from above and there is only one way to edit the position records. When finished with this routine, press {Q} to return to the main program menu.

PARAMETERS DESCRIPTION

There are six record types stored in the parameter file which is associated with the Control program. Most of these values can be changed by the user, but some are under program control only and cannot be changed. Engineering units for each of the parameters will be given under each description, but if the conversion equations have not been specified, the program recognizes all values as voltages. The following is a list of the parameters in each record and a brief description of what they do:

POSITION RECORD. There are 18 cart positions supported by the Control program. The chamber which is hooked up to the position is described by the following information:

Cart number. This is the number on the chamber itself.

Hood volume. This is the volume under the enclosure in liters.

Cart type. The type of cart; soil or hydroponic.

Air sampling. This is an indicator which is set on if you wish the chamber connected to the air sampling system (see pages 18-19) and is set off otherwise. Note that there are two indicators which must be on for air sampling to occur, this one and the Air Interrupt indicator which is set in the main menu (see the description for the current program values on page 15 and the description in the main menu section, page 5).

Regular sampling. Another indicator which allows you to select/deselect an individual cart from the main measurement system. Note that the Regular interrupt indicator (page 5) must also be on before any measurements can be taken. See the description of the measurement system on pages 17-18.

Maximum/Minimum allowable pH. This sets the allowable range for the pH. Then, during the measurement cycle, if the pH is too low, base is added; if too high, acid is added. The parameter is saved in pH units (0-14).

Time to add acid/base. If the pH does fall out of range during a measurement, the program will add acid or base (depending on which is needed) for the specified amount of time. Expressed in seconds.

Transpired solution dump. The pot which collects transpired solution can only hold a certain amount. This tells the program when to empty the pot. Expressed in mls.

Hydroponic solution add. This specifies the level for adding hydroponic solution. For soil carts, the level must exceed this value, and for hydroponic carts, the level must fall below this value. Expressed as mls.

Amount of solution to add. Tells the program the number of pump pulses of solution to add to the chamber when it needs solution.

Optimum CO_2 level. The program keeps the CO_2 level under the chamber hood at a constant level when the air sampling system is on (see pages 18-19). The optimum level is in parts per million (ppm).

Range of CO_2. Specifies the necessary accuracy for CO_2 control. The program will control CO_2 at the optimum level +/- this range. Expressed as ppm.

Conversion equations. Each measurement which is taken off the cart is taken from an electronic probe. These probes can be calibrated and a fourth order polynomial equation used to transform the voltages into engineering units. There is one equation saved for each measured value on each cart.

Light boundary. Specifies the boundary between lights on and lights off. Expressed as micromoles/sec/square meters.

Old or New air sampling system. The latching CO_2 flow controller is only found on later model carts. Carts which were made at an earlier time have a more crude method of CO_2 control.

Number of plants. This is used for calculations which determine rates per plant.

Current CO_2 flow rate. The CO_2 measurement system accesses this value for determining the CO_2 injection rate to apply to the cart. The rate is saved as ppm/min.

SWITCHES RECORD. Switches are used by the computer for such things as adding hydroponic solution to the carts, adding base or acid, and turning on the electrical conductivity switch. The switches record only holds the name of each switch. This is especially useful in the test devices section when you wish to turn a certain switch on or off.

CHANNELS RECORD. Each measurement made on each cart is made by reading an electrical probe or channel. The following is stored for each channel:

Name. The name of the channel can be 20 characters or less. If there are no characters in the name, the channel is automatically disconnected.

Scale. The voltage read off each channel is automatically scaled with this value to produce a number which is within a specified range. The user can select one of nine scales.

Maximum/Minimum graphics scale. Used during the display data routine. This specifies the maximum and minimum values to use on the y-axis while graphing data.

ERRORS RECORD. Since the computer is constantly monitoring each cart, it can recognize if a measurement falls outside a predetermined range. This can help avoid undesirable or even dangerous conditions in the laboratory. If this does occur, the computer will use the telephone to alert one of the laboratory technicians of the situation (see the section on Telephone operation, pages 28-29). The following information is stored in the error record:

Ranges. Each channel has a maximum and minimum value stored. If the computer detects a measurement outside this range, an error condition is flagged.

Time of last error. If an error occurs, the time is recorded by the computer.

Position of last error. The computer also records the position where the error occurred.

Channel of last error. And it pinpoints the channel which had the error.

Telephone calling on/off. This is an indicator; set on, the computer will access a phone list and call the people on the list until a connection is made. If the indicator is off, the phone will not be used.

Interval between phone calls. If the computer does not detect an answer when it is calling, it will call the next person on the list after this interval has passed.

MISCELLANEOUS RECORD. This is one of the most important pieces of information which the Control program uses. The following is stored in the miscellaneous record:

Air sample rate. This is the interval between air sample conditions (see the section on air sampling, pages 18-19). In seconds.

Sample rate. The interval between the regular measurement series. When this interval has been exceeded, the program initiates a measurement series.

Transition period length. In minutes. This defines the period of time it takes the plants to equilibrate between day and night.

Cart with light sensor. Only one position has the light sensor, this defines which it is.

Missing data flag. Missing data values are described on pages d-19-d-21 in the Statpro Database User's guide.

CO_2 flow rate. In liters/second. This is used in the CO_2 injection scheme. Used to determine the length of time needed to inject CO_2 to bring the CO_2 concentration in the chamber up to the optimum level.

Relative period to display. Control can display any period of statistics which has been collected. 0 will display the period of statistics currently being collected. Negative numbers will display that number of periods previous to the current period.

Position to display. Due to space limitations on the monitor screen, the program can display one position, all channels for statistics, or it can display one channel, all positions. This defines the position to display.

Channel to display. See the description for postion to display, above. This defines the channel to display in the one channel, all positions mode for statistics display.

Number of standard gas measurements to skip. Each time the gas cycle loops, the computer will take a measurement of a CO_2 standard and a N_2 zero gas. If it is not necessary to take this measurement each time a loop is made, this number can be set to some value greater than zero.

CURRENT PROGRAM VALUES RECORD. These are values which the program must remember when it is necessary to chain to another program in the Control system.

Current state of the switches. The switches will either be on or off. When the computer returns the system to Control, the switches will be flipped on or off, depending on their state before chaining occurred. These values cannot be changed by the user; except in the Test Devices routine (page 27).

Solution added today. One of the values saved in the workfile is the amount of solution added to each chamber during the day (24-hour period). The program must remember this value in order to keep accurate track of the solution added. These values cannot be changed by the user.

Last light reading. Normal CO_2 flow into each chamber is turned off when the lights are off. Since the air system is separate from the regular measurement system, this value, which reveals whether the lights are on or off, must be available while the air system routines are being executed.

Last gas measurement. Measured data values are stored on the datafile during the normal measurement routine. Since gas measurements are made at another time, they are stored here until the normal measurement routine is called. The CO_2 measurement, humidity, and time are stored. These values cannot be changed by the user.

Time of last gas measurement. Used for adjusting the normal CO_2 injection into each cart. These times are used to calculate the elapsed time between the current measurement series and the last measurement series. These values cannot be changed by the user.

Current state of the air cycle. There are two states, purging air lines with outside air, and filling the air lines with CO_2 from the chamber. These two states, in conjunction with the current chamber being accessed in the air system, define the state of the air cycle.

Current air status display. This is the message which is displayed on the main menu which reflects the current state of the air cycle. This value cannot be changed by the user.

Chamber in the air cycle. The air cycle works with only one chamber at a time. This value defines which chamber is being accessed. The value represents the position number of the chamber. If the value is -2, then N_2 zero gas is being measured, if the value is -1, the CO_2 standard is being measured.

Air measurement system indicator. The main control which toggles the air measurement system.

Measurement system indicator. This is the main control which toggles the regular measurement series.

Telephone answering indicator. Control can be configured to reroute input and output over the telephone. If telephone answering is on, the computer will answer the phone if it rings, and will send and receive information over the phone. If this indicator is off, the telephone is ignored.

Modem in use indicator. Certain delay routines must be implemented if the modem is being used. If the modem is being used, this indicator is on.

Statistics collection indicator. When this indicator is on, statistics will be collected. Note that if the statistics workfile is full, statistics collection will be turned off.

Regular collection indicator. This is analagous to the statistics collection indicator.

Workfile error indicator. If data collection has been terminated because of an I/O error, the program will beep every 5 seconds to alert the lab technicians that an error has occurred.

Screen number. This is the number of the main menu being displayed. 1 is the regular main menu. 2 is the statistics menu which displays one position and all channels. 3 is the menu which displays one channel and all positions.

Current period. There are four periods in the day for the purpose of calculating statistics. These are day, day to night transition, night, night to day transition. The current period is also stored in the statistics workfile.

Last period. This is the period preceding the current statistical period. This is used to determine whether the current statistical data should be saved or removed.

Number of standard gas measurements skipped. This value keeps track of the standard gas measurements skipped. As soon as this value exceeds the number of skips specified (see number of standard gas measurements to skip parameter), the standard gas and the zero gas will be measured.

Amount of transpired solution collected during the current statistical period. One value is saved for each cart which is on line. These values are used to determine the rate of transpiration in each chamber. These values cannot be changed by the user.

MEASUREMENT SERIES

The Control program is designed to automatically take channel measurements on each chamber every time a specified amount of time has elapsed (the sample rate in the miscellaneous record of the parameter file, see page 14) or when the Measure option has been selected from the main menu. When either of these has occurred, the program sequentially accesses each on-line position and measures each channel on each chamber, flagging error conditions and summing statistics as it goes. When measurements on all the chambers are complete, the flagged error conditions are handled by the addition of acid/base, the addition of solution, or by dumping the transpired solution.

The measurements are displayed on the screen while the chamber is being measured.

There are two options which are available during the measurement process:

{esc}: Rejects the displayed measurements and does not store them on the datafile. Any measurements taken prior to pressing {esc} are already saved on the workfile.

{W} : Waits, the data currently being displayed will remain on the screen until {space} is pressed. This allows the opportunity to view the measurements being collected before the program continues on to the next chamber in the series.

The following are relevant parameters for the measurement system. See the parameters description section, pages 11-17.

Position record:

Cart type
Regular sampling
Maximum/Minimum allowable pH
Time to add acid/base
Transpired solution dump
Hydroponic solution add
Amount of solution to add
Conversion equations
Light boundary
Number of plants

Channels record: The channels must have a name and the appropriate scale before measurements will be made on them.

Miscellaneous record:

- Sample rate
- Transition period length (for statistics collection)
- Cart with light sensor

Current program values record:

- Solution added today
- Measurement system indicator
- Regular workfile full
- Amount of transpired solution collected

AIR MEASUREMENTS

Part of the measurement system of Control is the system designed to monitor and regulate the CO_2 concentration within each chamber. The actual measurements are made by an Infra-red gas analyzer (IRGA) which is connected by air hoses to each cart and to the computer as one of the input channels. Since the movement of air through the air loops takes much longer than simple electronic measurements, it was necessary to develop a measurement cycle which is different than the regular measurement cycle.

Starting at the first chamber in the system, the air lines are purged with air, filled with air from the chamber, and the measurement is made. If there is a discrepancy between the desired and the actual CO_2 concentration in the chamber, the CO_2 input rate is adjusted, and CO_2 is injected into the chamber if necessary. When one chamber has been measured, the next chamber in the system is measured in the same way.

After the last chamber in the system has been measured, the program will do one of two things: it will either start over at the first chamber in the system, or it will measure a CO_2 standard gas and an N_2 standard zero gas for calibration purposes. Once the standard gases have been measured, the program starts over at the first chamber in the system.

The number of standard gas measurements can be regulated by editing the number of standard gas measurements to skip parameter in the miscellaneous record (page 15). For example, if the number of measurements to skip was 1, the air cycle loop would skip one standard measurement, i.e. it would take standard measurements every other air cycle.

The following are relevant parameters for the air system. See the parameters description section, pages 11-17, for a description on how each parameter affects the air system.

Position record:

Hood volume
Air sampling
Optimum CO_2 level
Range of CO_2
Conversion equations
Light boundary
Old or New air sampling system

Channels record:

Channel 11 is the CO_2 channel, channel 12 is the water vapor channel and channel 13 is the mass flow (CO_2 input rate) channel. Each of these channels should be appropriately named and the channel scale should be set to +5.0v.

Miscellaneous record:

Air sample rate
CO_2 flow rate
Number of standard gas measurements to skip

Current values record:

Last light reading
Last gas measurement
Time of last gas measurement
Current state of the air cycle
Current air status display
Chamber in the air cycle

Air measurement system indicator
Number of standard gas measurements skipped

DATA COLLECTION

As the Control program is measuring the chambers, it simultaneously saves the measurements in a workfile. The format of the workfile was described in the Output section, page 1. The user can collect data on all the carts, some of the carts, or data collection can be turned off completely. Data collection is turned off by the program if the workfile becomes full.

The following parameters are used to turn workfile storage on or off:

Position record:

Regular sampling indicator

Current values record:

Measurement system indicator

Both of these parameters must be set on for measurements to take place. If the Current values measurement system indicator is on, measurements will be made on those positions which have the position record regular sampling indicator on, but will not be made on those positions which have that parameter off.

The following are the measurements stored in each of the 36 fields of the workfile:

Chan #	Name
1.	Temperature
2.	Temperature
3.	Temperature
4.	Temperature
5.	empty
6.	pH
7.	Light measurement
8.	Solution level
9.	Transpired solution level
10.	empty
11.	Electrical conductivity
12.	CO_2 concentration
13.	Water vapor
14.	CO_2 flow control reading
15.	Total solution added since midnight
16.	Transpired solution per plant (during the current period)
17.	CO_2 assimilation rate
18.	Position cart
19.	Date
20.	Time
21.	Time of CO_2 measurement
22.	N_2 measurement (when taken)
23.	CO_2 standard measurement (when taken)
24.	Time of standard gas measurement (when taken)
25.	Elapsed time
26.	Number of plants
27-36.	empty

STATISTICS COLLECTION AND DISPLAY

COLLECTION. Along with storing raw data, the Control program is also capable of collecting statistical information on each of the measured channels. The information on each channel-position pair is stored as a record in a workfile which is separate from the regular workfile (but with the same format). The mean, standard deviation, coefficient of variation, maximum and minimum observations, as well as the formula of a line fitted to the data by linear regression and the various sums used to calculate the statistics are stored on the workfile. Statistics are collected during day and night periods. There are also transition periods between the day and night periods; statistics are calculated during these periods but are not saved. Statistics collection can be turned on or off; and when on, statistics are collected on all the carts which are currently being measured.

The following parameter turns statistics collection on/off:

Current values record:

Statistics collection indicator. See page 16.

Define the length of the transition between the day and night periods with the following parameter:

Miscellaneous record:

Transition period length. See page 14.

Appendix 1

The following are the values stored in each of the 36 fields of the statistics workfile:

1. empty
2. Channel
3. First date of statistical period
4. First time of statistical period
5. empty
6. empty
7. Number of observations
8. Sum of values
9. Sum of values squared
10. Maximum value
11. Minimum value
12. Period number (1=night, 2=transition, 3=day, 4=transition)
13. Sum of elapsed time
14. Sum of elapsed times squared
15. Sum of elapsed time * value
16. Mean
17. Standard deviation
18. Coefficient of variation
19. Slope of fitted regression equation
20. Position.cart
21. Date
22. Time
23. Y axis intercept for fitted regression equation
24. r
25. Standard error of fitted regression equation

26-36. empty

DISPLAY. The accumulating statistics can be displayed in place of the main menu if that is desired. There are two types of statistical displays; the first displays the statistics for one position and all channels:

```
Time:  [ 9:39:36]  Date:  [15 Jan 85]  Memory:  [ 7090]   Record:  [ 6315]
Air system [purging]  Stats  [+]  Data [+]  Regular [+]  Air [+]  Phone [-]
Position [ 7]  Cart [11]                    Date [  1.1585]  Period [  day]

     Chan            N       Mean/a          CV/r         Max/SE        Min
[condensor temp] [   53] [      4.35 ] [      1.80 ] [   4.45 ] [    4.20 ]
[air chamber te] [   53] [     22.90 ] [      6.22 ] [  24.06 ] [   20.18 ]
[solution temp.] [   53] [     23.37 ] [      3.54 ] [ 24.25  ] [   21.86 ]
[air room temp.] [   53] [     22.05 ] [      3.00 ] [  22.57 ] [   20.65 ]
[              ] [    0] [           ] [           ] [        ] [         ]
[pH            ] [   53] [      5.16 ] [      0.95 ] [   5.26 ] [    5.12 ]
[light         ] [   53] [      1.74 ] [      2.41 ] [   1.78 ] [    1.65 ]
[hydropcnic sol] [   53] [      6.54 ] [      0.51 ] [   6.62 ] [    6.47 ]
[transpired sol] [   53] [      0.03 ] [     34.16 ] [   0.05 ] [    0.02 ]
[oxygen        ] [   53] [      0.00 ] [     14.44 ] [   0.00 ] [    0.00 ]
[elec. conducti] [   53] [     81.60 ] [      4.41 ] [  86.04 ] [   76.53 ]
[CO2 concentrat] [   53] [    386.95 ] [     15.12 ] [ 484.44 ] [  345.11 ]
[water vapor   ] [   53] [     16.16 ] [     16.83 ] [  18.11 ] [    6.86 ]
[              ] [    0] [           ] [           ] [        ] [         ]
[              ] [    0] [           ] [           ] [        ] [         ]
[              ] [    0] [           ] [           ] [        ] [         ]
[soln added tod] [   53] [    161.89 ] [     45.00 ] [ 300.00 ] [   90.00 ]
[transp. / plan] [   53] [      0.00 ] [      0.00 ] [   0.00 ] [    0.00 ]
[photosynth / p] [    0] [           ] [           ] [        ] [         ]
<--/--> Select Screen;   R)eturn to Pascal;   Error [ 0 ]
```

The second screen is similar, it displays all positions for one channel:

```
Time:  [ 9:39:36]  Date:  [15 Jan 85]  Memory:  [ 7090]  Record:  [ 6315]
Air system [ purging] Stats  [+]  Data [+]  Regular [+]  Air [+]  Phone [-]
Channel [ 0]   Name [condensor temp.     ]  Date [15 Jan 85]  Period [  day]
      Pos          N        Mean/a         CV/r         Max/SE        Min
[            0] [  53] [     4.34 ] [     1.79 ] [     4.43 ] [     4.19 ]
[            1] [    ] [          ] [          ] [          ] [          ]
[            2] [   0] [          ] [          ] [          ] [          ]
[            3] [   0] [          ] [          ] [          ] [          ]
[            4] [  53] [     4.40 ] [     1.81 ] [     4.51 ] [     4.30 ]
[            5] [   0] [          ] [          ] [          ] [          ]
[            6] [   0] [          ] [          ] [          ] [          ]
[            7] [  53] [     4.34 ] [     1.80 ] [     4.45 ] [     4.20 ]
[            8] [   0] [          ] [          ] [          ] [          ]
[            9] [   0] [          ] [          ] [          ] [          ]
[           10] [   0] [          ] [          ] [          ] [          ]
[           11] [   0] [          ] [          ] [          ] [          ]
[           12] [   0] [          ] [          ] [          ] [          ]
[           13] [  53] [     4.32 ] [     1.90 ] [     4.44 ] [     4.21 ]
[           14] [   0] [          ] [          ] [          ] [          ]
[           15] [   0] [          ] [          ] [          ] [          ]
[           16] [   0] [          ] [          ] [          ] [          ]
[           17] [   0] [          ] [          ] [          ] [          ]

<--/--> Select Screen;   R)eturn to Pascal;   Error [ 0 ]
```

See the section on the main menu, page 4, for a description of how to select these displays.

The position (for one position-all channels) and the channel (for one channel-all positions) to display are selected by editing the position to display parameter and the channel to display parameter in the miscellaneous record. These parameters are described on page 14.

EXAMINE

Selecting option {F} from the main menu suspends the Control program and begins the Examine program. This program graphs data on the screen. Data can be selected by channel-position, record number or time of collection. When the Examine program starts, the following menu will be displayed:

```
Examine: ?)  S)pecify  R)ecords  P)lot  F)lip  G)raph  Q)uit
```

?) Displays a screen which describes each of the commands shown.

S)pecify. This command allows the set-up of a list of the channel-position pairs to plot. There is a maximum of 6 channel-position pairs which can be plotted at any time (more than 6 causes an unreadable graph). Press {S} to see the following:

Specify: ?) N)ew A)dd D)elete V)iew <esc>

?) displays a screen describing each of these commands.

N)ew. This command clears a previously constructed list.

A)dd. A list of the on-line positions and a list of the channels is displayed on the right of the screen. The current list of channel-position pairs to plot is also displayed. Enter in the position, {return}, followed by the channel, {return}, of the pair to appear on the plot. Press {esc} when finished entering pairs. The current list and the Specify menu will then be redisplayed.

D)elete. Allows deletion of specific channel-position pairs from the list. The current list of pairs will be displayed. Press {return} to keep a pair in the list and press {D} to delete a pair. When finished, the new channel-position pair list and the Specify menu will be displayed.

V)iew. Displays the current list of channel-position pairs.

<esc> returns to the Examine menu.

R)ecords. This command allows selection of the record range to plot and to select the y axis scaling factor for each channel. The range can be selected by choosing record numbers or by choosing time and date. Pressing {R} reveals the following:

Records: ?) T)ime N)umber S)cale <esc>

?) displays a screen describing each of these commands.

T)ime. Displays a screen with the range of dates and times currently in the workfile. Move to the date or time to change and enter in the new time or date. Press {esc} when finished. The program will now take some time to locate the exact positions of these times and dates in the workfile. Patience.

N)umber. Displays a screen with the range of record numbers in the file. Move to the record number to change; change it. Press {esc} when finished. The program will now locate these records in the file. Note that this way of selecting the record range to plot is considerably faster than selecting by time and date.

S)cale. This allows the y axis scaling to be set for each of the channels. See page 13 for the description on the Maximum/Minimum graphics scale parameter in the Channels record. Each channel has its own y axis scale; move to the channel to edit, and enter in either the maximum, which will be the top of the y axis for this channel, or the minimum, which will be the bottom of the y axis. Note that each channel has its own scaling factor. This means that when the plot is produced, there is actually a different y axis scale for each of the channels plotted.

<esc> returns to the Examine menu.

P)lot. The program will now display the graph and begin plotting the list of position-channel pairs between the record range which have been specified. When plotting is finished, the program switches to a text screen and displays a report of the plot, explaining which channel-position goes with which symbol, and it displays the record range by time/date and record number. The graph can be displayed as long as desired (until the next graph is plotted) by using the F)lip command. To interrupt plotting at any time, press {esc}.

F)lip. This command allows transfer from the currently plotted graph to the report screen and back. The graph will stay in memory as long as another graph has not been plotted over it, so the graph which was plotted last can be viewed at any time by pressing {F}.

G)raph. Graphs can be saved onto disk for later viewing, or can be printed on an Epson printer using this utility. Note that saving a graph onto the disk, and then retrieving it later is actually quicker than replotting all of the data. The report screen is not duplicated, however. Press {G} to see:

```
Graph:  ?)  D)ump  S)ave to disk  R)etrieve from disk  <esc>
```

?) displays a screen describing these commands.

D)ump. If an Epson printer is connected to the computer, the graph which is currently in memory can be printed by pressing {D}. Note that the printer should be hooked up. A caption can be printed on the graph. If everything is ok, the graph will take about 5 minutes to print.

S)ave to disk. There may be times when a graph should be saved from the screen onto a disk. To do this, it is necessary to have on hand one of the special 'PHOTOS' disks (see the Statpro graphics user guide). When prompted, insert the photos disk into disk drive #5. Enter a file number between 1 and 17 for saving the graph and press return. Press {return} without entering a number to abort the procedure. The actual save will take a couple of seconds. The Graph menu will be redisplayed.

R)etrieve from disk. This is the opposite of the S)ave command. Insert the photos disk, enter the number of the photo to read, {return}, and the graph will be loaded into memory. Use the F)lip command to view the graph or go back to the text screen.

<esc> returns to the Examine menu.

Q)uit ends the Examine program and returns to the Main menu of the Control program.

TEST DEVICES

Pressing {H} from the main menu displays the test devices menu:

```
-- Test Devices --

a) Clock
b) Switches
c) Cart enable
d) Air enable
e) CO2 flow controller
f) A/D measurements
g) Irrigate a cart

Enter choice: [ ]
```

These commands allow selection of routines which can test and calibrate the hardware associated with the system.

a. Clock. This routine returns the current time and date as defined by the California Computer Systems (CCS) clock card inside the Apple. To change the date and time on the clock, use the Set Clock option, {K}, on the main menu. Press {esc} to return to the test devices menu.

b. Power switching unit. This routine allows the control switches to be manually turned on and off. A screen of each switch's name and current status is displayed. Move to the switch to change, and press {<} to turn the switch off, or {>} to turn the switch on. Press {esc} to return to the test devices menu.

c. Cart enable. This allows the linkage between any position and the computer to be established. Enabling a position is necessary to regulate and take measurements from the cart connected to it. Enter the position to enable, {return}. The position selected will be enabled, all other positions will be disabled. To abort this routine, press {esc}.

d. Air enable. This option allows the air loop connection between the IRGA and any position to be made. The air pump (a switch, used for pumping the air from the chamber) is not affected. Enter the position to enable, {return}. To abort this routine, press {esc}.

e. CO_2 flow controller. This option allows the CO_2 input rate to be changed on any on-line chamber. Enter the position number to change, and then enter the voltage to send to the flow controller, a number between 0.1 and 5.0, which is the relative rate of CO_2 input to the chamber. To abort this routine, press {esc} at any time.

f. Analog / Digital measurements. This option allows measurement on one particular position-channel combination. Enter the position and channel to measure, {return}. The numbers displayed are the voltage measured from that position-channel combination, and the measurement which has been transformed into engineering units based on the calibration equations. Press {esc} when finished or press {+} to enter a new position-channel combination.

g. Irrigate a cart. This option allows any amount of hydroponic solution to be added to a cart. Enter the destination for the solution (a position), {return}, and the number of pumps, {return}. Each pump cycle is approximately 1 ml. The pumping starts immediately. The test devices menu will then be redisplayed. Press {esc} at any time to stop pumping.

Q)uit redisplays the Main CONTROL menu.

TELEPHONE OPERATION

There are two functions which the telephone performs for CONTROL: alerting the laboratory technicians of an error or potentially dangerous situation in the laboratory, and receiving telephone calls for the purpose of displaying the CONTROL output at a remote location.

ALERT. This option is enabled by setting the telephone alert calling on/off indicator in the Errors record (see description, page 14). Errors on the carts are flagged during the measurement series according to the ranges parameter in the Errors record (page 13). If a measurement is found to be outside the range set for it, CONTROL accesses a text file named PHNUMBERS. TEXT on the disk drive, and loads in a list of phone numbers to call.

The phone number file is a standard Pascal text file which can be modified or created by the Apple Pascal text editor. The program reads this file line for line until the end is reached or 4 valid phone numbers are read in. The structure of the telephone number file is as follows:

```
1 7574670    Craig McFarlane's office
| | |             |
| | |        anything here and beyond is ignored, so you may comment.
| | |
| | phone number(may be up to 11 chars)
| |
| always blank
|
special code (see below)
```

1 = Mon thru Fri, 8:00am to 5:00pm.
2 = Mon thru Thurs, 5:00pm to 8:00am, Fri, 5:00pm to midnight.
3 = Sat or Sun anytime.
4 = Mon thru Fri anytime.
5 = Mon thru Fri days, Sat or Sun anytime.
6 = Mon thru Fri nights, Sat or Sun anytime.
* = anyday of the week and anytime.

Anything else in the first column is ignored and the phone number on that line is not used.

A call is attempted and the program will wait about a minute for a signal from the other end. Signal by whistling a rising scale into the phone, the louder the better. If no signal is received then the program proceeds to the next number on the list. Each number (if no answer occurs on any of the previous attempts) will be called 3 times. If no answer is received after 3 tries, the program will attempt this again after a period of time established by the interval between phone calls parameter in the Errors record (page 14).

RECEIVE. Using the proper connection program, a remote computer can call in to CONTROL and have all screen output from CONTROL sent across the phone. Access to CONTROL can be denied at the laboratory end by setting the telephone answering indicator in the Current program values record off (see page 16). All program operations occur normally while telephone communication is in progress. Hanging up at the remote end causes CONTROL to hang up at its end and re-route screen output normally.

SELECT/MERGE PROGRAM

This is a special program which allows transfer of workfile records from the Corvus hard disk to floppy disks. All, or just some of the records can be transferred. Selection is based upon the record numbers, dates, times, positions and carts in the workfile. Records can be appended to an existing workfile on a floppy disk, or a new workfile can be started. Apple Pascal formatted disks must be used, but they need not be blank. If the disk contains one or more files, a prompt will ask for confirmation before writing over the disk. The program writes a DEFAULTS file on the disk, as well as the WKEYFILE, WLABELS, and WORKFILE files. A floppy disk can hold up to 940 workfile records.

The Select/Merge Program main menu consists of three options:

T(ransfer workfile using default options
C(hange existing transfer options
R(eturn to Control Program

T(ransfer workfile using default options. This option initiates the transfer of all the workfile records on source volume #11 to destination volume #9, creating a new workfile. If the C(hange existing transfer options selection is made, the T(ransfer prompt will change upon returning to the main menu. It will now read:

T(ransfer workfile using existing options

Choosing this option will begin the transfer of selected workfile records based upon the current status of the transfer options.

C(hange existing transfer options. This choice allows the current transfer options to be changed. A change options menu is provided:

```
P(osition Selection
C(art Selection
O(ther options [Drives, dates, times, record #'s]
R(eturn to main menu
```

P(osition Selection. Select any of the 18 positions [0..17] to transfer. A position is selected by typing its number and pressing {return}. {+} selects all positions. The default is all positions selected.

C(art Selection. Select any of the 100 carts [0..99]. A cart is selected by typing its number and pressing {return}. {+} selects all carts. The default is all carts selected.

O(ther options [Drives, dates, times, record #'s]. Provides a full screen of choices to be made. The following shows the options, with the defaults appearing in [ ]'s:

```
Source drive               [ 11 ]
Destination drive          [ 9 ]
Starting record number     [ 1 ]
Ending record number       [ Last record in file ]
Starting date              [ First date in file ]
Ending date                [ Last date in file
Starting time              [ 00.0000 ]
Ending time                [ 23.5959 ]
Appending to old workfile  [ No ]
```

Use {<--} and {-->} to move up and down through the options and change them.

R(eturn to main menu. Returns to the Select/Merge main menu.

R(eturn to Control Program. Leave the Select/Merge Program and return to the Control Program.

APPENDIX 2

CONTROL PROGRAM: SOURCE CODE AND DESCRIPTION

Source Text Files for the Computer Programs Used to Manage Equipment and Data in the Plant Exposure Laboratory

by

Mike McDowell

Toxic and Hazardous Materials Branch
Corvallis Environmental Research Laboratory
200 S.W. 35th Street
Corvallis, Oregon 97333

CORVALLIS ENVIRONMENTAL RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CORVALLIS, OREGON 97333

ACKNOWLEDGEMENTS

I would like to thank the following for their contributions to CONTROL: Allen Todd, who wrote the assembly language routines and the first Pascal code for CONTROL, Robert Clark for his telephone alert procedures, Brian Martin for writing the Select Program, CALL A.P.P.L.E.'s Ron DeGroat and Chris Wilson for the keyboard input and modem routines used in CONTROL, and Tom Swan for many routines from his PDBS system. I would also like to thank my colleagues at EPA, especially Craig Mc Farlane for his program ideas and Tom Pfleeger for his uncanny ability to find bugs.

DISCLAIMER

The information in this document has been funded wholly or in part by the U.S. Environmental Protection Agency. It has been subjected to the Agency's peer and administrative reviews and has been approved for publication as an EPA document. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

Appendix 2

CONTENTS

Page

Introduction 2-1
Units 2-2
  External Procedures 2-3
  Controlstuff 2-15
  Workstuff 2-41
Main Program 2-49
Auxiliary Programs 2-89

INTRODUCTION

This appendix is a listing of all the program elements which make up the CONTROL system. The assembly procedures are written in 6502 assembly language for the Apple Pascal assembler and the Pascal routines are written in U.C.S.D. Pascal for the Apple Pascal 1.1 or 1.2 compiler. Please see the Apple Pascal Language reference manual and the Apple Pascal Operating System manual for descriptions on library units and segment procedures, and any other information about about Apple Pascal which may be necessary.

There are four programs which, chained together, comprise CONTROL. These are: The CONTROL program itself (control.code), the Define Parameters program (define.code), the Examine Program (examine.code), and the Select/Merge program (select.code). The following library units supplied by Apple are used: Turtlegraphics (Examine), and Chainstuff (all programs).

UNITS

There are two regular library units for CONTROL:

CONTROLSTUFF. Contains the constants, types and variables used by CONTROL, keyboard input routines, screen display routines, CONTROL parameter file routines, Hayes Micromodem routines, clock routines and some other handy utilities. The procedures and functions in controlstuff are used in nearly all phases of the CONTROL program. The text files are:

a) CTRLSTUFF.TEXT (See page 2-15)
b) CTRLSTUFF1.TEXT (See page 2-24)

WORKSTUFF. This unit contains the constants, types, variables and routines needed to take measurements on the plant chambers, and handle data storage. This unit is used only in the data collection and environmental control segment procedures. The text files are:

a) DRIVERS.TEXT (See page 2-3)
b) CLOCK&KEY.TEXT (See page 2-11)
c) WRKSTUFF.TEXT (See page 2-41)

To use these units, follow these steps:

1) Compile CTRLSTUFF.TEXT.
2) Assemble DRIVERS.TEXT and CLOCK&KEY.TEXT.
3) Compile WRKSTUFF.TEXT.
4) After CONTROL has been compiled, link CONTROL with the resulting codefiles from steps 1-3.

Appendix 2

```
        .TITLE  "ASSEMBLER DRIVERS FOR EPA CONTROL PROGRAM"
;
;
;*********************************************************************
;
;                          DRIVER ROUTINES FOR
;
;             INTERACTIVE STRUCTURES DI09/AI16 POWER CONTROLLER
;           INTERACTIVE STRUCTURES AI13 ANALOG/DIGITAL CONVERTER
;
;                                  BY
;
;                            ALLEN W. TODD
;                     TODD MICROCOMPUTER CONSULTANTS
;                      2711 N. W. GIBSON HILL ROAD
;                        ALBANY, OREGON   97321
;                           (503) 926-2986
;
;       REVISION 1.5    October 16, 1985
;                       by Mike McDowell
;
;*********************************************************************
;
;
;
;=====================================================================
;-------------- GLOBAL DEFINITIONS -----------------------------------
;
RETURN  .EQU    0                       ;PASCAL RETURN ADDRESS
DI09    .EQU    0C500                   ;BASE ADDRESS FOR DI09 CARD IN SLOT 5
;                                        set address with $C(slot)00
AI13    .EQU    0C090                   ;BASE ADDRESS FOR AI13 IN SLOT 1
IRQVECT .EQU    0FFFE                   ;6502 IRQ VECTOR ADDRESS
;
;
        .MACRO  POPINT                  ;POP INTEGER ARGUMENT FROM STACK
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM
;
        .MACRO  PUSHINT                 ;PUSHES INTEGER ARGUMENT ONTO STACK
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM
;=====================================================================
        .PROC   INITDRIVERS
;
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;       PROCEDURE Initdrivers;
;
;       PERFORMS ALL NECCESSARY INITIALIZATION FOR DRIVERS
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;
;-------------- LOCAL EQUATES ----------------------------------------
;
DDR0    .EQU    DI09+3                  ;6522 DATA DIRECTION REGISTERS
DDR1    .EQU    DI09+2
DDR2    .EQU    DI09+13
DDR3    .EQU    DI09+12
PORT0   .EQU    DI09+1                  ;6522 I/O REGISTERS
PORT1   .EQU    DI09+0
PORT2   .EQU    DI09+11
```

```
PORT3   .EQU    DI09+10
PCRO    .EQU    DI09+0C            ;PERIPHERAL CONTROL REG. 0
ACR1    .EQU    DI09+1B            ;6522 AUXILLARY CONTROL REGISTER 1
IFR1    .EQU    DI09+1D            ;6522 INTERUPT FLAG REGISTER 1
IER1    .EQU    DI09+1E            ;6522 INTERUPT ENABLE REGISTER 1
T2LLO   .EQU    DI09+14            ;6522 TIMER 2 LATCH LSB
T2LHI   .EQU    DI09+15            ;6522 TIMER 2 LATCH MSB
T3LLO   .EQU    DI09+18            ;6522 TIMER 3 LATCH LSB
T3CHI   .EQU    DI09+19            ;6522 TIMER 3 COUNTER MSB
TIMEO   .EQU    013F8              ;TIMER 2 COUNT VALUE
        .PUBLIC SWITCHSTATE,INTERUPT
        .DEF    OLDVECT
        .REF    TIME1              ;TIMER #3 COUNT
;
;----------------------------------------------------------------------
;
;
        LDA     #0FF
        STA     DDRO               ;PROGRAM PORT 0 FOR OUTPUT
        STA     PORTO              ;TURN ON ALL BITS, ALL SWITCHES OFF
        STA     DDR1               ;PROGRAM PORT 1 FOR OUTPUT
        STA     PORT1              ;TURN ON ALL BITS, ALL SWITCHES OFF
        STA     DDR2               ;PROGRAM PORT 2 FOR OUTPUT
        STA     PORT2              ;TURN ON ALL PORT 2 BITS
        LDA     #0BF
        STA     DDR3               ;PORT 3; DL30 AS INPUT, REMAINDER AS OUTPUT
        LDA     #3F
        STA     PORT3              ;TURN ON ALL BUT 30 & 31
        LDA     #0CC
        STA     PCRO               ;SET CL1 AND CL3 LOW
        LDA     #0CE
        STA     PCRO               ;SET CL1 HIGH
        LDA     #0CC
        STA     PCRO               ;SET CL1 AND CL3 LOW
        LDA     #0
        STA     SWITCHSTATE        ;INITIALIZE SOFTCOPY OF PORTS 0 & 1
        STA     SWITCHSTATE+1
        STA     INTERUPT+1         ;INIT PASCAL INTERUPT FLAG MSB
        SEI                        ;NO INTERUPTS YET
        STA     IFR1               ;MAKE SURE
        LDA     #0A0
        STA     IER1               ;ENABLE INTERUPTS FROM TIMER 3
        LDA     #0E0
        STA     ACR1               ;PROGRAM TIMERS 2 AND 3
        LDA     TO
        STA     T2LLO              ;INIT TIMER 2
        LDA     TO+1
        STA     T2LHI              ;AND START COUNTING
        LDA     IRQVECT            ;PRESERVE EXSISTING IRQ VECTOR
        STA     OLDVECT
        LDA     IRQVECT+1
        STA     OLDVECT+1
        LDA     SERVICE            ;SET UP NEW IRQ VECTOR
        STA     IRQVECT
        LDA     SERVICE+1
        STA     IRQVECT+1
        RTS
;
;
TO      .WORD   TIMEO              ;TIMER 2 CYCLE COUNT
OLDVECT .WORD   0                  ;STORAGE FOR PREVIOUS IRQ VECTOR
SERVICE .WORD   IRQSTUF            ;ADDRESS OF IRQ SERVICE ROUTINE
;
;
;<<<<<<<<<<<<<<<<<<<< INTERUPT SERVICE ROUTINE >>>>>>>>>>>>>>>>>>>>>
;
IRQSTUF PHA                        ;SAVE A REGISTER
        TXA
        PHA                        ;SAVE X REGISTER
        TYA
        PHA                     ;SAVE Y REGISTER
        LDA     #0
        STA     INTERUPT        ;RESET PASCAL INTERUPT FLAG
        STA     IFR1            ;CLEAR INTERUPT FLAG
        LDA     TIME1           ;RESTART TIMER 3
        STA     T3LLO
        LDA     TIME1+1
        STA     T3CHI
;
;----   DO INTERUPT SPECIFIC STUFF HERE ----------
;
        LDY     #050            ;JUST BEEP THE SPEAKER FOR NOW
BEEP    LDA     #0C
        SEC                     ;DELAY, OLD MONITOR WAIT ROUTINE
MWAIT1  PHA
MWAIT2  SBC     #1
        BNE     MWAIT2
        PLA
        SBC     #1
        BNE     MWAIT1
        LDA     0C030            ;TOGGLE SPEAKER
        DEY
        BNE     BEEP
        SEC                     ;SET PASCAL INTERUPT FLAG
        ROL     INTERUPT
        PLA                     ;RESTORE Y REGISTER
        TAY
        PLA                     ;RESTORE X REGISTER
        TAX
        PLA                     ;RESTORE A REGISTER
        RTI                     ;RESUME PROCESSING
;======================================================================
        .PROC   SETTIMER,1
;
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;       PROCEDURE Settimer(Time: INTEGER);
;
;       SETS TIMED INTERUPT INTERVAL, Time (10 mSEC INCREMENTS)
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;
;-------------- LOCAL EQUATES -----------------------------------------
;
T3LLO   .EQU    DI09+18         ;6522 TIMER 3 LATCH LSB
T3CHI   .EQU    DI09+19         ;6522 TIMER 3 COUNTER MSB
IFR1    .EQU    DI09+1D         ;6522 INTERUPT FLAG REGISTER 1
        .DEF    TIME1           ;COUNT FOR TIMER #3
;
;---------------------------------------------------------------------
;
;
;
        POPINT  RETURN
        PLA                     ;GET TIME INTERVAL, LSB
        STA     TIME1           ;SAVE FOR IRQ SERVICE ROUTINE
        STA     T3LLO           ;PUT IN LOW ORDER LATCH
        PLA                     ;GET TIME INTERVAL, MSB
        STA     TIME1+1         ;SAVE FOR IRQ SERVICE ROUTINE
        STA     T3CHI           ;START TIMER #3
        LDA     #0
        STA     IFR1            ;CLEAR INTERUPT FLAG
        CLI                     ;ENABLE INTERUPTS
        PUSHINT RETURN
        RTS
;
;
TIME1   .WORD   0               ;STORAGE FOR TIMER #3 COUNT
```

```
;
;===================================================================
        .PROC   DO_SWITCH,2
;
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;       PROCEDURE do_switch( channel: integer; state: onoff );
;
;       TURNS ON OR OFF ONE CHANNEL OF DI09/UI16
;       CHANNEL CONTAINS THE CHANNEL TO BE ALTERED
;       IF STATE = 0 THEN CHANNEL IS SWITCHED OFF
;       IF STATE = 1 THEN CHANNEL IS SWITCHED ON
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;
;----------------------- LOCAL EQUATES -----------------------------
;
STATE   .EQU    2
CHAN    .EQU    3
MASK    .EQU    4
        .PUBLIC SWITCHSTATE     ;SOFT COPY OF 16. SWITCH BITS
;
;-------------------------------------------------------------------
;
;
        POPINT  RETURN          ;SAVE PASCAL RETURN ADDRESS
        PLA                     ;GET STATE
        STA     STATE
        PLA                     ;DISCARD MSB
        PLA                     ;GET CHANNEL NUMBER
        STA     CHAN
        PLA                     ;DISCARD MSB
        LDA     CHAN            ;SELECT REGISTER SPECIFIC OUTPUT ROUTINE
        CMP     #8.
        BCC     REG0
        CMP     #16.
        BCC     REG1
DONE    PUSHINT RETURN          ;DONE, RESTORE PASCAL RETURN ADDRESS
        RTS                     ;AND GO HOME
;
;       CHANNELS 0..7
;
REG0    LDY     #1              ;INDEX TO PROPER DATA REGISTER
        TAX
        LDA     PWROF2,X        ;SELECT BIT TO ALTER
        STA     MASK
        LDX     #0              ;INDEX INTO SWITCHSTATE ARRAY
UPDATE  JSR     SETCLR          ;SWITCH SELECTED BIT OF SWITCHSTATE ARRAY
        EOR     #0FF            ;COMPLEMENT BIT STATES FOR NEGATIVE LOGIC
        STA     DI09,Y          ;SEND OUT BYTE
        JMP     DONE
;
;       CHANNELS 8..15
;
REG1    LDY     #0              ;INDEX TO CORRECT DATA REGISTER
        SEC
        SBC     #8              ;CALCULATE RELATIVE BIT
        TAX
        LDA     PWROF2,X        ;SELECT BIT TO ALTER
        STA     MASK
        LDX     #1              ;INDEX INTO SWITCHSTATE ARRAY
        JMP     UPDATE
;
;       SET OR CLEAR THE PROPER BIT
;
SETCLR  LSR     STATE           ;SHIFT OUT STATE BIT
        BCS     SETBIT          ;SET IT
CLRBIT  JSR     SETBIT          ;FIRST SET THE BIT
```

```
        LDA     MASK
        EOR     #0FF            ;COMPLEMENT MASK
        STA     MASK
        LDA     SWITCHSTATE,X
        AND     MASK            ;NOW CLEAR THE BIT
        STA     SWITCHSTATE,X   ;UPDATE BIT ARRAY
        RTS
SETBIT  LDA     SWITCHSTATE,X   ;GET CURRENT BIT STATE
        ORA     MASK            ;SET THE BIT
        STA     SWITCHSTATE,X   ;UPDATE STATE ARRAY
        RTS
;
;       TABLE OF POWERS OF 2
;
PWROF2  .BYTE   1,2,4,8,10,20,40,80
;
;
;=====================================================================
        .PROC   DO_CARTSELECT,1
;
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;               PROCEDURE do_cart_select(cartnum: INTEGER);
;
;       Selects one cart 'cartnum' to be the active cart for control
;       and measurements.  All other carts are disabled.  Cartnum may be
;       turned off.
;
;       DL 24 ---> DL29
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;--------------- LOCAL EQUATES ----------------------------------------
;
PORT3   .EQU    DI09+10         ;ADDRESS OF OUTPUT PORT 3
PCR0    .EQU    DI09+0C         ;PERIPHERAL CONTROL REG. 0
;
;---------------------------------------------------------------------
;
;
        POPINT RETURN           ;SAVE PASCAL RETURN ADDRESS
        PLA                     ;GET LSB CARTNUM
        AND     #3F             ;MASK OFF UNUSED BITS
        STA     PORT3           ;TURN ON CORRECT CART
        PLA                     ;DISCARD MSB CARTNUM
        LDA     #0CE
        STA     PCR0            ;TOGGLE CL1
        LDA     #0CC
        STA     PCR0
        LDA     #0FF            ;CLEAR PORTS
        STA     PORT3
        PUSHINT RETURN          ;RESTORE PASCAL RETURN ADDRESS
        RTS
;====================================================================
        .PROC DO_GASSELECT,1
;
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;
;                  PROCEDURE do_gas_select(valve: INTEGER);
;
;       Turns on any gas valve for sampling exposure chamber atmosphere.
;       Valve may be in the range 0..255.  If Valve=0 then all valves are
;       turned off.
;
;       DL 16 ---> DL23
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;--------------- LOCAL EQUATES ------------------------------------------
;
PORT2   .EQU    DI09+11          ;ADDRESS OF OUTPUT PORT 2
PCR0    .EQU    DI09+0C          ;PERIPHERAL CONTROL REG. 0
;
;-----------------------------------------------------------------------
;
;
        POPINT  RETURN           ;SAVE PASCAL RETURN ADDRESS
        PLA                      ;GET LSB VALVE
        STA     PORT2            ;TURN ON CORRECT VALVE
        PLA                      ;DISCARD MSB
        LDA     #0CE
        STA     PCR0             ;TOGGLE CL1
        LDA     #0CC
        STA     PCR0
        LDA     #0FF             ;CLEAR PORTS
        STA     PORT2
        PUSHINT RETURN           ;RESTORE PASCAL RETURN ADDRESS
        RTS
;=======================================================================
        .PROC DAC,1
;
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;
;                   PROCEDURE DAC(VALUE: INTEGER);
;
;       SENDS VALUE TO DIGITAL TO ANALOG CONVERTER ON THE CURRENTLY
;       SELECTED CART.  THE OUPUT NUMBER MUST BE IN THE RANGE 0..511.
;
;       DL 16 ---> DL25 ARE USED
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;--------------- LOCAL EQUATES ------------------------------------------
;
PORT2   .EQU    DI09+11          ;ADDRESS OF OUTPUT PORT 2
PORT3   .EQU    DI09+10          ;ADDRESS OF OUTPUT PORT 3
PCR0    .EQU    DI09+0C          ;PERIPHERAL CONTROL REG. 0
VALUE   .EQU    2                ;TEMP
;
;-----------------------------------------------------------------------
;
;
        POPINT  RETURN           ;SAVE PASCAL RETURN ADDRESS
        POPINT  VALUE            ;GET VALUE TO SEND
        LDA     VALUE            ;LSB VALVE
        STA     PORT2            ;SEND LOWER BYTE
;  SEND WORD WITH LOAD BIT HIGH
        LDA     VALUE+1          ;MSB VALUE
        AND     #1               ;MASK OFF UNUSED BITS
        ORA     #2               ;SET BIT 1
        STA     PORT3            ;SEND MSB
        JSR     LATCH2           ;TOGGLE CL3
;  SEND WORD WITH LOAD BIT LOW
        LDA     VALUE+1          ;MSB VALUE
        AND     #1               ;MASK OFF UNUSED BITS
        STA     PORT3            ;SEND MSB
        JSR     LATCH2           ;TOGGLE CL3
;  SEND WORD WITH LOAD BIT HIGH
        LDA     VALUE+1          ;MSB VALUE
        AND     #1               ;MASK OFF UNUSED BITS
        ORA     #2               ;SET BIT 1
        STA     PORT3            ;SEND MSB
        JSR     LATCH2           ;TOGGLE CL3
        LDA     #0FF             ;CLEAR PORTS
        STA     PORT2
        STA     PORT3
        PUSHINT RETURN          ;RESTORE PASCAL RETURN ADDRESS
        RTS
;
LATCH2  LDA     #0EC
        STA     PCR0
        LDA     #0CC
        STA     PCR0
        RTS
;
;=================================================================

        .FUNC DO_ANALOG, 2
;
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;       FUNCTION do_analog(channel: INTEGER; range: voltrange): INTEGER;
;
;               READS AI13 ANALOG TO DIGITAL CONVERTER
;       SELECTING INPUT FROM CHANNEL AND SETS APPROPRIATE VOLTAGE RANGE
;            RETURNS INTEGER VALUE (0...4095) OF VOLTAGE READ
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;
;
;------------- LOCAL EQUATES ----------------------------------------
;
CHAN    .EQU    2               ;TEMP STORAGE FOR CHANNEL NUM
FLAGS   .EQU    3               ;BIPOLAR AND X10 RANGE FLAGS
;
;-------------------------------------------------------------------
;
;
        POPINT  RETURN          ;SAVE PASCAL RETURN ADDR
        PLA
        PLA                     ;DISCARD 4 BYTES STACK BIAS
        PLA
        PLA
        PLA                     ;GET RANGE CODE
        AND     #0F             ;MASK OFF HIGH NIBBLE
        ASL     A
        ASL     A               ;MULTIPLY RANGE CODE BY 16.
        ASL     A
        ASL     A
        TAX                     ;SAVE RANGE CODE IN X REG
        ASL     A               ;ISOLATE BIPOLAR & X10 BITS
        STA     FLAGS           ;SAVE AS BITS 6 & 7 OF FLAG BYTE
        PLA                     ;DISCARD MSB RANGE
        PLA                     ;GET CHANNEL NUMBER
        AND     #0F             ;MASK OFF HIGH NIBBLE
        STA     CHAN
        PLA                     ;DISCARD MSB CHAN
        TXA                     ;RECOVER RANGE CODE
        CLC
        ADC     CHAN            ;ADD TO CHANNEL NUMBER
        BIT     FLAGS           ;CHECK FOR X10 RANGE
        BVC     READA2D         ;NOT X10 RANGE, READ A/D
        STA     AI13            ;X10 RANGE, START DUMMY CONVERSION
        LDX     #8
DELAY   DEX                     ;WAIT FOR PRE-AMP TO STABILIZE
        BPL     DELAY
READA2D STA     AI13            ;START CONVERSION
        PHA
        PLA                     ;WAIT FOR MSB CONVERSION
        LDA     AI13+1          ;READ MSB
        AND     #0F             ;MASK OFF HIGH NIBBLE
        NOP
        LDX     AI13            ;READ LSB
```

```
DONE    PHA                     ;RETURN MSB VOLTAGE VALUE
        TXA
        PHA                     ;RETURN MSB VOLTAGE VALUE
        PUSHINT RETURN          ;RESTORE PASCAL RETURN ADDR
        RTS
;======================================================================
        .PROC RESTORE
;
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;       PROCEDURE Restore;
;
;       RESTORES NORMAL INTERUPT VECTORS
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;
;
;------------- LOCAL EQUATES ------------------------------------------
;
IER1    .EQU    DIO9+1E         ;6522 INTERUPT ENABLE REGISTER 1
        .REF    OLDVECT         ;COPY OF ORIGINAL IRQ VECTORS
;
;----------------------------------------------------------------------
;
;
        SEI                     ;MAKE SURE INTERUPTS OFF
        LDA     #80
        STA     IER1            ;DISABLE INTERPTS FROM 6522
        LDA     OLDVECT         ;RESTORE VECTORS
        STA     IRQVECT
        LDA     OLDVECT+1
        STA     IRQVECT+1
        RTS
;======================================================================
        .PROC   ENABLE
;
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;       PROCEDURE Enable;
;
;       ENABLES INTERUPT PROCESSING
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;
        CLI
        RTS
;======================================================================
        .PROC   DISABLE
;
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;       PROCEDURE Disable;
;
;       DISABLES INTERUPT PROCESSING
;
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;
        SEI
        RTS
;======================================================================
        .END
        .TITLE  "SOME COMMON ASSEMBLER ROUTINES"
;
;
;*******************************************************************
;
;                  SOME GENERAL PURPOSE ASSEMBLY ROUTINES
;
;
;                       FUNCTION readtime: INTEGER;
;
;                       FUNCTION keypress: BOOLEAN;
;
;  ================================================================
;
;                                BY
;
;                           ALLEN W. TODD
;                     TODD MICROCOMPUTER CONSULTANTS
;                      2711 N. W. GIBSON HILL ROAD
;                        ALBANY, OREGON   97321
;                           (503) 926-2986
;
;       October 5, 1985
;
;*******************************************************************
;
;
;
;==================================================================
;-------------- GLOBAL DEFINITIONS ---------------------------------------
;
RETURN  .EQU    0                   ;PASCAL RETURN ADDRESS
CLOCK   .EQU    0C0C0               ;BASE ADDRESS FOR CLOCK IN SLOT 4
;                                   ;calculate with $C0(8+slot)0
;
        .MACRO  POPINT              ;POP INTEGER ARGUMENT FROM STACK
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM
;
        .MACRO  PUSHINT             ;PUSHES INTEGER ARGUMENT ONTO STACK
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM
;
```

```
;========================================================================
;
;
;
        .FUNC   READTIME
;
;
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;                            FUNCTION readtime: INTEGER;
;
;                                     Reads
;               California Computer Systems 7424 Clock/Calendar Card
;
;               Returns time (24 hr format) as an integer and updates
;               Pascal global date record to reflect the current date
;
;                             date:    PACKED RECORD
;                                         mo:   1..12;
;                                         day:  1..31;
;                                         year: 0..99;
;                                      END;
;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;-------------- LOCAL EQUATES -------------------------------------------
;
TIME    .EQU    2               ;2 BYTE TIME TEMPORARY
TEMP    .EQU    4               ;TEMPORARIES
MASK    .EQU    5
SECS    .EQU    31              ;ADDRESS LATCH BYTES IMMEDIATE DEFINITIONS
MINS    .EQU    33
HRS     .EQU    35
DAYS    .EQU    38
MONS    .EQU    3A
YRS     .EQU    3C
        .PUBLIC SECONDS,DATE
;
;------------------------------------------------------------------------
;
        POPINT  RETURN          ;SAVE PASCAL RETURN ADDRESS
        PLA
        PLA                     ;DISCARD 4 BYTES STACK BIAS
        PLA
        PLA
        LDA     #0
        STA     SECONDS+1       ;ZERO OUT MSBS
        STA     TIME+1
        LDY     #7              ;SET UP BIT MASK
        LDA     #SECS
        JSR     GETDATA         ;READ SECONDS
        STA     SECONDS
        LDY     #3              ;SET UP BIT MASK
        LDA     #HRS
        JSR     GETDATA         ;READ HOURS
        STA     TIME
        JSR     MULT10          ;MULTIPLY HOURS BY 100
        JSR     MULT10
        LDY     #7              ;SET UP BIT MASK
        LDA     #MINS
        JSR     GETDATA         ;READ MINUTES
        CLC
        ADC     TIME            ;ADD MINUTES TO HRS * 100
        STA     TIME
        LDA     TIME+1
        ADC     #0
        STA     TIME+1
        LDY     #0F             ;BIT MASK
        LDA     #YRS
        JSR     GETDATA         ;READ YEARS
        STA     DATE+1
        LDY     #3              ;BIT MASK
        STA     MASK
        LDA     #DAYS
        JSR     GETDATA         ;READ DAYS
        ASL     A
        ASL     A               ;FORMAT FOR DATE RECORD
        ASL     A
        ASL     A
        ROL     DATE+1          ;SHIFT MS BIT DAY INTO LS BIT DATE+1
        STA     DATE
        LDA     #MONS
        JSR     GETDATA         ;READ MONTHS
        ORA     DATE            ;COMBINE MONTH AND DAY FOR DATE RECORD
        STA     DATE
        LDA     #0              ;DISABLE CLOCK I/O
        STA     CLOCK+1
        PUSHINT TIME            ;RETURN CURRENT TIME
        PUSHINT RETURN          ;RESTORE PASCAL RETURN ADDRESS
        RTS                     ;TAKE 'ER HOME
;
;------ READ THE CLOCK. SHIFT 2 BCD DIGITS INTO 8 BIT BINARY FORMAT --------
;
GETDATA TAX                     ;SAVE ADDRESS BYTE
        STY     MASK
        STA     CLOCK+1         ;LATCH X 10 TIME DIGIT
        LDA     CLOCK           ;READ TIME DIGIT
        AND     MASK
        PHA                     ;BCD TO 8 BIT BINARY MULTIPLY BY 10 ROUTINE
        ASL     A               ;MULTIPLY BY 2
        STA     TEMP
        PLA                     ;RECOVER MULTIPLICAND
        ASL     A
        ASL     A               ;MULTIBLY BY 8
        ASL     A
        ADC     TEMP            ;RESULT = TIMES2 + TIMES8
        STA     TEMP            ;SAVE RESULT IN TEMP
        DEX
        TXA
        STA     CLOCK+1         ;LATCH X 1 TIME DIGIT
        LDA     CLOCK           ;GET THE TIME DIGIT
        AND     #0F
        CLC
        ADC     TEMP            ;ADD IN X 10 DIGIT. RETURN RESULT IN A REG.
        RTS
;
;       MULTIPLY TIME BY 10
;
MULT10  LDA     TIME
        ASL     A               ;MULTIPLY LSB BY 2
        PHP                     ;SAVE CARRY BIT
        ASL     TIME
        ROL     TIME+1          ;MULTIPLY BY 8
        ASL     TIME
        ROL     TIME+1
        ASL     TIME
        ROL     TIME+1
        CLC
        ADC     TIME            ;ADD PARTIAL PRODUCTS
        STA     TIME
        LDA     TIME+1
        ADC     #0
        PLP                     ;RECOVER CARRY
        ADC     #0
        STA     TIME+1
        RTS
;
```

```
;
        .FUNC   KEYPRESS,0
;
;**********************************************************************
;*                                                                    *
;*                 FUNCTION keypress: BOOLEAN;                        *
;*                 ==========================                         *
;*                                                                    *
;*      Reads the Apple keyboard or the external console keyboard     *
;*              returning true if a key has been pressed              *
;*                         false otherwise                            *
;*                                                                    *
;**********************************************************************
;
;
;
;---------LOCAL EQUATES -----------------------------------------------
;
RPTR    .EQU    0BF18   ;PASCAL TYPE AHEAD BUFFER READ POINTER
WPTR    .EQU    0BF19   ;PASCAL TYPE AHEAD BUFFER WRITE POINTER
GOCONCK .EQU    0BF0A   ;PASCAL SYSTEM ROUTINE TO CALL CONCK
;
;----------------------------------------------------------------------
;
;
;
        POPINT  RETURN          ;SAVE PASCAL RETURN ADDRESSS
        PLA
        PLA                     ;DISCARD FOUR BYTES STACK BIAS
        PLA
        PLA
        JSR     GOCONCK         ;GO TO CONCK ROUTINE IN BIOS
BUFRCK  LDA     WPTR            ;CHECK TYPE AHEAD BUFFER
        CMP     RPTR
        BNE     DONE            ;CHAR IN BUFFER, CARRY IS SET
        CLC                     ;NO CHAR IN BUFFER, CLEAR CARRY
DONE    LDA     #0
        PHA                     ;RETURN MSB BOOLEAN
        ROL     A               ;SHIFT IN BOOLEAN BIT
        PHA                     ;RETURN LSB BOOLEAN
        PUSHINT RETURN          ;RESTORE PASCAL RETURN ADDRESS
        RTS
;
;
        .END
```

```
{ L ctrl.lst.text}
UNIT control_stuff;

{---------------------------------------------------}
{  Control_stuff unit - this is the basic           }
{  unit required by the Control program.  The       }
{  procedures in this unit are used in nearly       }
{  all portions of the program.  Contributers       }
{  to this unit are:                                }
{  Ron DeGroat -        Keyboard input routines }
{  Chris Wilson -       Modem routines              }
{  Allen Todd -         Assembly routines           }
{  Tom Swan -           PDBS routines               }
{                                                   }
{  version  7 Oct 85    0930              -mkm-     }
{---------------------------------------------------}


INTERFACE


CONST

{------------------------------}
{ Miscellaneous constants      }
{------------------------------}

max_field              =      36;              { max # fields in workfile  }
max_per_disk           =      maxint;          { max records per file      }
error_line             =      20;              { line for error display    }
page_size              =      1024;            { for FILE pages, buf size  }
missing                =      -999;            { flag for missing data     }
min_mem                =      2048;            { Minimum words memory      }
                                               { maintained                }

wrk_extension          =      ':WORKFILE';     { workfile name             }
key_extension          =      ':WKEYFILE';     { keyfile name              }
lab_extension          =      ':WLABELS';      { labelfile name            }
prm_extension          =      ':PARAMETERS';   { parameter file name       }
month_names            =      '???JanFebMarAprMayJunJulAugSepOctNovDec';

maxposition            =      18;      { max # of positions (1..18)        }
maxcart                =      99;      { max # of carts (1..99)            }
maxchan                =      16;      { max # of channels (1..16)         }
maxswitch              =      16;      { max # of switches (1..16)         }
samplecount            =      20;      { mean of 20 samples => reading     }
noerror                =      0;       { no error currently                }
loginerror             =      1;       { no response to login request      }
chanerror              =      2;       { error in reading a channel        }
lagtime                =      20.0;    { wait time after co2 inject        }
period                 =      5;       { measurement period in minutes     }
maxstatrec             =      20521;   { maximum statistics record number  }
                                       { approx. 30 days of storage when   }
                                       { system is at maximum capacity.    }
crit_light             =      0.5;     { lights off value, for transition  }
n2_std                 =      -1;      { code for nitrogen standard        }
co2_std                =      0;       { code for co2 standard             }
maxreal                =      9999E32;{ very large real number            }
minreal                =      -999E32;{ small real number                 }
off_cart               =      64;      { number which disables carts       }

{------------------------------}
{ Parameter file block numbers  }
{------------------------------}

                                       { block 0 is empty                  }
                                       { blks 1-maxposition have cart recs }
mirec                  =      19;      { misc parameter block number       }
```

```
curec                         =      20;    { current values param block number }

{-------------------------------}
{ Screen number constants       }
{-------------------------------}

meas_screen                   =      10;    { screen to display during measures }

{-------------------------------}
{ Switches constants            }
{-------------------------------}

vent                          =       1;    { vent to the outside                }
airswitch                     =       2;    { outside air                        }
transoff                      =       3;    { turn off transpired solution dump  }
transswitch                   =       4;    { transpired soln drain              }
co2on                         =       5;    { turn co2flow back on               }
co2off                        =       6;    { turn co2flow off                   }
elecswitch                    =       7;    { electrical conductivity            }
n2gas                         =       8;    { std nitrogen sample                }
acidswitch                    =       9;    { add acid                           }
airpump                       =      10;    { draw air from the chambers         }
squirt                        =      11;    { co2 inject                         }
cartsoln                      =      12;    { soln on cart                       }
solenoid                      =      13;    { needed for soln pump               }
solnswitch                    =      14;    { add solution                       }
co2stdgas                     =      15;    { std co2 sample                     }
baseswitch                    =      16;    { add base                           }

{-------------------------------}
{  Channels, Statpro fields     }
{  constants.                   }
{-------------------------------}

                                            { chans 1-5 are temperature         }
                                            { thermistors.                      }
ph_chan                       =       6;    { pH data channel                   }
lite_chan                     =       7;    { light measurement                 }
soln_chan                     =       8;    { solution level                    }
tran_chan                     =       9;    { transpired soln data              }
                                            { 10 is empty                       }
elec_chan                     =      11;    { electrical conductivity           }
co2_chan                      =      12;    { co2 concentration                 }
h2o_chan                      =      13;    { water vapor                       }
co2_flow                      =      14;    { co2 flow control reading          }
                                            { 15-16 are empty                   }
tot_soln                      =      17;    { solution added data               }
tran_plnt                     =      18;    { transpired solution data / plant  }
                                            { note - this is the total          }
                                            { transpired solution during the    }
                                            { current period.                   }
phot_plnt                     =      19;    { photosynth rate estimate / plant  }
                                            { note - these last two could       }
                                            { easily be eliminated - the        }
                                            { calculations for them could be    }
                                            { performed by any post-processing  }
                                            { program.  If these channels are   }
                                            { ever needed for something impt,   }
                                            { get rid of these ( 18, 19 ) first.}
pos_cart                      =      20;    { format, pos.cart                  }
date_fld                      =      21;    { todays date, format -> MM.DDYY    }
time_fld                      =      22;    { time of measurement, HH.MMSS      }
co2_time                      =      23;    { time of co2 measurement           }
n2_fld                        =      24;    { N2 std gas measurement            }
co2_fld                       =      25;    { CO2 standard gas measurements     }
std_time                      =      26;    { time of standard gas measure      }
elap_time                     =      27;    { field for the elapsed time        }
plnt_fld                      =      28;    { field for the number of plants    }
```

```
tran_rate              =     29;    { transpired solution rate - calc.  }
                                    { 30-36 are empty                    }

{------------------------------}
{  Statistics workfile consts   }
{------------------------------}

chanfield              =      1;    { channel of measurement             }
firstdatefield         =      2;    { first date in the period           }
firsttimefield         =      3;    { first time in this period          }
nfield                 =      7;    { number of observations             }
sumy                   =      8;    { sum of the measured values         }
sumysqr                =      9;    { sum of squared measureds           }
maximum_field          =     10;    { maximum measured value             }
minimum_field          =     11;    { minimum measured value             }
periodfield            =     12;    { period code number                 }
sumx                   =     13;    { sum, elapsed times for linear reg }
sumxsqr                =     14;    { sum of squared elapseds            }
sumxy                  =     15;    { sum of value * elapsed             }
meanfield              =     16;    { calculated mean of measured value }
sdfield                =     17;    { calculated std deviation           }
cvfield                =     18;    { calculated coeff of variation      }
slopefield             =     19;    { slope of line - linear reg         }
lastdatefield          =     21;    { last date in this period           }
lasttimefield          =     22;    { last time in this period           }
yintfield              =     23;    { y intercept of line - linear reg  }
rfield                 =     24;    { how well line fits data            }
errfield               =     25;    { standard error of line             }

TYPE

{------------------------------}
{ miscellaneous types           }
{------------------------------}

path_name              =     string[ 23 ];  { for file names            }
error_type             =     0..87;         { for reporting errors      }
ucsd_date              =     PACKED RECORD  { internal date structure   }
                               mo    :      0..12;
                               day   :      0..31;
                               yr    :      0..100;
                             END;
setofchar              =     SET OF char;
page_ptr               =     ^ucsd_page;
ucsd_page              =     PACKED ARRAY[ 1..page_size ] OF char;
stack_ptr              =     ^integer;
onoff                  =     ( off, on );
voltrange              =     ( plus5v, plus1v, plus500mv, plus100mv,
                               dual5v, dual1v, dual500mv, dual100mv,
                               disconnect );

{------------------------------}
{ Modem types                   }
{------------------------------}

baudrate               =     ( low,         { 110 baud                  }
                               high );      { 300 baud                  }
mode                   =     ( answer, originate );

{------------------------------}
{ Statpro workfile structures   }
{------------------------------}

workrec                =     RECORD
                               data         : ARRAY[ 1..max_field ] OF
                               real;
                               dflag        : char;
                             END;
```

```
keyrec                  =      RECORD
                                 title         : string[ 40 ];
                                 rtotal,       { total number of records    }
                                 fieldnum,     { # of recognized fields     }
                                 fnum          { # of marked records        }
                                               : integer;
                               END;
string9                 =      string[9];
labelrec                =      RECORD
                                 clabel      : string9;
                               END;
label_array             =      ARRAY[ 0..max_field ] OF string9;
mark_type               =      ( marked, unmarked, both );

{------------------------------}
{  Miscellaneous types, used   }
{  here and there.             }
{------------------------------}

gastype          =      RECORD          { gas measurement storage record    }
                          time,         { time of measurement               }
                          photodata,    { photosynthetic rate calculation   }
                          h2odata,      { water vapor measurement           }
                          co2data       { co2 measurement                   }
                                        :  real;
                        END;

airtype          =                      { conditions of the air system      }
                        ( fill,         { filling lines from chamber        }
                          purge,        { purging with air                  }
                          gasmeas );    { measuring co2 concentration       }

period_type      =                      { periods of the day ( 24 hrs )     }
                        ( tran_to_nite, { transition between day and night  }
                          night,        { night                             }
                          tran_to_day,  { transition between night and day  }
                          day );        { day                               }

time_type        =      RECORD          { note when something occurred      }
                          lastsecs,     { the seconds                       }
                          lasttime      { the time in HHMM format           }
                                        :  integer;
                          lastdate      { the date in ucsd date format      }
                                        :  ucsd_date;
                        END;

trans_array      =      ARRAY[ 1..6 ]   { calculate transpiration rate      }
                          OF RECORD     { there is one array per position   }
                            value,      { amount of soln collected / plant  }
                            etime       { elapsed time since prev. measure  }
                                        : real;
                            abs_time    { the date and time of this measure }
                                        : time_type;
                          END;


{------------------------------------------------------------------------}
{  Parameter file types.  The parameter file holds all the information    }
{  which is required for the Control program.  Each block holds one of the }
{  record types as specified below.  In order to save the overhead memory  }
{  required when a file is declared, I will use the file ( f ) which is    }
{  declared in the statpro_stuff unit, and will use the same logic that is }
{  used for the load, update, and initialize procedures for the keyfile    }
{  and the labels file.                                                    }
{                                                                          }
{    Block              Contents                                           }
{    -----              --------                                           }
{    1 - maxposition    cartp record for each position.                    }
{                         these can be loaded from separate cart files which }
{                         are of type cart. Not permanent. The proper record }
{                         is loaded only when needed.                        }
{                                                                            }
{    maxposition + 1      miscp record - permanent.  Includes old errors,    }
{                         miscellaneous and channels records.                }
{                                                                            }
{    maxposition + 2      currp - permanent.  The current values record.     }
{----------------------------------------------------------------------------}


{------------------------------}
{  Cart record structures      }
{------------------------------}

phrec          =     RECORD          { pH information, in pos record      }
                       upperph,      { maximum allowable pH in chamber    }
                       lowerph       { minimum - in pH units, 1-14        }
                                     : real;
                       timetoadd     { time to add acid/base - seconds    }
                                     : integer;
                     END;

solnrec        =     RECORD          { soln information, in posn record   }
                       transpired    { level high, dump transp. soln.     }
                                     :  real;
                       solution      { level low, add hydroponic soln.    }
                                     :  real;
                       amount        { # pumps of solution to add         }
                                     : integer;
                     END;

cartco2        =     RECORD          { co2 information, in posn record    }
                       optimum,      { desired co2 level - in ppm         }
                       span          { acceptable range of co2 - in ppm   }
                                     : real;
                     END;

equations      =                     { 4th order polynomial equation      }
                                     { parameters to convert voltages to  }
                                     { real scientific values. In         }
                                     { position record.                   }
                     ARRAY[ 1..5, 1..maxchan ] OF real;

kindcart       =                     { the type of cart.  In posn. rec.   }
                     ( hydroponic,   { hydroponic style carts             }
                       soil,         { soil/Las Vegas style carts         }
                       extra1,       { future use                         }
                       extra2 );     { future use                         }

cartinfo       =     RECORD          { position information               }
                       cartnum,      { cart num. as opposed to posn. #    }
                       num_plants,   { number of plants in cart           }
                       lite_pos,     { pos for this cart's lite measure   }
                       added_today,  { amount of solution added today     }
                       base_ptr      { 1st rec for stats in workfile      }
                                     : integer;
                       aironline,    { take air measurements ?            }
                       online,       { take regular measurements ?        }
                       newair,       { using the new air system ?         }
                       dosed         { has the chamber been dosed ?       }
                                     : boolean;
                       hoodvol,      { hood air volume, in liters         }
                       lightinfo,    { boundary between lights on/off     }
                                     { units: micromoles/sec/meters sqd.  }
                       currrate,     { currnt co2 flow rate, newair only  }
                       lastco2,      { last co2 measurement - in ppm      }
                       tottranspired,{ total transpired soln collected    }
```

```
                    last_lite      { last light measure for this cart  }
                                   : real;
                    carttype       { the type of cart at this position }
                                   : kindcart;
                    phinfo         { pH information                    }
                                   : phrec;
                    solninfo       { hydroponic/transpired soln info   }
                                   : solnrec;
                    co2info        { co2 information                   }
                                   : cartco2;
                    conversions    { volts -> scientific values        }
                                   : equations;
                    trans_rate     { calculate the transpiration rate  }
                                   : trans_array;
                    dose_time,     { time of dosing                    }
                    rate_time      { time of last co2 measure - for    }
                                   { rate calculation.                 }
                                   : time_type;
                  END;  { cartinfo }

{------------------------------}
{ Miscellaneous record         }
{------------------------------}

chaninfo       =                   { channel voltage scaling factors   }
                  ARRAY[ 1..maxchan ] OF voltrange;

errorrec       =  RECORD           { record of environmental problems  }
                    errorranges    { legal ranges, outside of this     }
                                   { range, an error condition exists  }
                                   : ARRAY[ 1..2, 1..maxchan ] OF real;
                    errortime,     { time-last alert phone call, HHMM  }
                    errorinterval, { interval between alert phone      }
                                   { calls, in minutes.                }
                    currerror,     { current error, see const decl.    }
                    errorchan,     { error at this channel             }
                    errorposition  { error at this position            }
                                   : integer;
                    callok         { ok to make alert phone calls ?    }
                                   : boolean;
                  END;

miscrec        =  RECORD           { record of misc. program info      }
                    flowrate       { co2 flow rate from bottle,        }
                                   { liters/second.                    }
                                   : real;
                    airsamprate,   { # seconds between air condition   }
                                   { changes, purge->fill->measure     }
                    samprate,      { # seconds between measurements    }
                    transperiod,   { transition period length, minutes }
                    relperiod,     { the relative period to display    }
                                   { for statistics display.  0 is the }
                                   { current period, -1 the previous   }
                                   { period, etc.                      }
                    pos,           { position to display, statistics   }
                    chan,          { channel to display, statistics    }
                    skip,          { number of standard gas (co2, n2)  }
                                   { measurments to skip.  Normally,   }
                                   { a measurement is made before the  }
                                   { first cart in the air measurement }
                                   { series.                           }
                    numpoints      { number of points to use when      }
                                   { calculating transpiration rate    }
                                   : integer;
                    channels
                                   : chaninfo;
                    errors
                                   : errorrec;
```

```
                          collect_stats { collect stats on these fields      }
                                        : ARRAY[ 1..max_field ] OF boolean;
                        END;

{-------------------------------}
{ Current values record         }
{-------------------------------}

currrec         =       RECORD          { current program state             }
                          switchvals    { state of the switches, on/off     }
                                        : ARRAY[ 1..maxswitch ] OF boolean;
                          gasarray      { set of last co2 measurements.     }
                                        { standard measurements are stored  }
                                        { in 0 for co2 and -1 for n2        }
                                        : ARRAY[ n2_std..maxposition ] OF
                                          gastype;
                          aircondition  { air condition to perform next     }
                                        : airtype;
                          gascart,      { current position for air system   }
                          skipcount     { the number of standard gas        }
                                        { measures skipped.                 }
                                        : integer;
                          airmeasure,   { perform air measurements ?        }
                          regmeasure,   { perform regular measurements ?    }
                          answerok,     { ok to answer phone, send data ?   }
                          stat_collect, { collect statistics ?              }
                          modemonline,  { is the modem in use ?             }
                          beep,         { beep when data file is full ?     }
                          data_collect  { ok to collect environmental data ?}
                                        : boolean;
                          screennum,    { current screen being displayed -  }
                                        { 1 = regular menu; 2 = stats, one  }
                                        { pos, all chans; 3 = stats, one    }
                                        { channel, all positions.           }
                          statvolume,   { volume for statistics workfile    }
                          workvolume    { volume for regular workfile       }
                                        : integer;
                          airstring     { current air system display string }
                                        : string[ 30 ];
                          currperiod,   { the current period                }
                          lastperiod    { the last period                   }
                                        : periodtype;
                        END;


VAR

ch,
ret,
space,
plus,
minus,
keyleft,
keyright,
keyup,
keydown,
esc,
keyetx                 :       char;

f                      :       FILE;   { file for miscellaneous i/o        }
infile,                                { file for remote input             }
outfile                                { file for remote output            }
                       :       interactive;
date                                   { current date                      }
                       :       ucsd_date;

labels                 :       label_array;
keys                   :       keyrec;
```

```
mark_char               :       ARRAY[ marked..unmarked ] OF char;

recnum,                                 { absolute record number            }
time,                                   { current time                      }
seconds,                                { current seconds                   }
lastsec,                                { secs, last time clock was read    }
samptime,                               { time, HHMM, of next measurement   }
sampseconds,                            { time, secs, of next measurement   }
airtime,                                { time, HHMM, of next air operation }
airseconds,                             { time, secs, of next air operation }
transtime                               { time transition period is ended   }
                :       integer;
cart                                    { Position record, from paramfile   }
                :       cartinfo;
misc                                    { Miscellaneous record, paramfile   }
                :       miscrec;
currvals                                { Current values record, paramfile  }
                :       currrec;
gas_pos,                                { table of positions to air measure }
pos_to_measure                          { table of positions to measure     }
                :       ARRAY[ 1..maxposition ] OF integer;
modem_in_use    :       boolean;        { modem being used                  }
exit_set,                               { chars allow exit from kybd funcs  }
cmd_set         :       setofchar;      { program defined commands          }

{-------------------------------}
{ error handling procedure      }
{-------------------------------}

PROCEDURE show_error( err: error_type );

{-------------------------------}
{ read labels procedure         }
{-------------------------------}

FUNCTION get_labels( volume: integer; VAR labels: label_array ): error_type;

{-------------------------------}
{ Parameter file routines       }
{-------------------------------}

FUNCTION load_parms( vol, block_num: integer ): error_type;
FUNCTION update_parms( vol, block_num: integer ): error_type;

{-------------------------------}
{ screen output procedures      }
{-------------------------------}

PROCEDURE evon;
PROCEDURE evoff;
PROCEDURE goright;
PROCEDURE goup;
PROCEDURE beep;
PROCEDURE clreoln;
PROCEDURE clreosc;
PROCEDURE clrscrn;
PROCEDURE goxy( x, y: integer );
PROCEDURE prompt( s: string );
PROCEDURE promptln( s: string );
PROCEDURE promptat( x, y: integer; s: string );
PROCEDURE prompt_real( r: real; width, decimals: integer );
PROCEDURE disp_screen( screen_num: integer );

{-------------------------------}
{ keyboard input procedures     }
{-------------------------------}
```

```
FUNCTION keypress: boolean;
FUNCTION getchar( okset: setofchar; echo, sensitive: boolean ): char;
FUNCTION finished( firstchar: boolean; ch: char; end_char_set: setofchar )
                                                               :boolean;
FUNCTION getnum( oldval, maxval, minval: real; ptok: boolean;
                                            VAR last_char: char ): real;
FUNCTION fpnum( fpstr: string ): real;
FUNCTION getinteger( default, maxval, minval: integer; VAR last_char: char )
                                                              : integer;

{-------------------------------}
{ various goodies               }
{-------------------------------}

FUNCTION readtime: integer;
FUNCTION verified( s: string ): boolean;
FUNCTION executed( e: error_type ): boolean;
FUNCTION elapsed_time( nowdate: ucsd_date; nowtime, nowsecs: integer;
                       thendate: ucsd_date; thentime, thensecs: integer ): real;
FUNCTION calc_secs( lasttime, lastsecs: integer): integer;

PROCEDURE perform( e: error_type );
PROCEDURE stopwatch( seconds: real );
PROCEDURE date_2_string( date: ucsd_date; VAR date_string: string );
PROCEDURE up_case( VAR ch: char );

{-------------------------------}
{ Modem procedures              }
{-------------------------------}

FUNCTION ringing: boolean;
FUNCTION carrier: boolean;
FUNCTION rcvrfull: boolean;

PROCEDURE initcomm( wordselect: integer );
PROCEDURE setmode( md: mode );
PROCEDURE enabletransmit;
PROCEDURE disabletransmit;
PROCEDURE pickup;
PROCEDURE unconnect;
PROCEDURE dial( number: string );
PROCEDURE answerthephone;


IMPLEMENTATION


{$I ctrlstuff1.text }


BEGIN   { control_stuff }
  initialize;
END.    { control_stuff }
```

```
{------------------------------}
{  ctrlstuff1.text             }
{                              }
{  The implementation section  }
{  of the control_stuff unit.  }
{                              }
{   7 Oct 85    0845    -mkm-  }
{------------------------------}

TYPE

crt_command     =
                       ( crt_evon,     { Enhanced video on }
                         crt_evoff,    { Enhanced video off }
                         crt_goright,  { Move non-destructively right }
                         crt_goup,     { Move cursor non-destructively up }
                         crt_bell,     { Ring bell }
                         crt_cleoln,   { Clear from cursor to end of line }
                         crt_cleosc,   { Clear to end of screen }
                         crt_clscrn ); { Clear entire screen }

crt_rec         =                      { Cathode Ray Tube (display) record }
                       RECORD
                         crt_n : 0 .. 4;
                         crt_ch : PACKED ARRAY[ 0 .. 3 ] OF char
                       END; { crt_rec }

word            =      PACKED ARRAY [0..1] OF char;

statusword      =      PACKED RECORD
                         filler1       : 0..127;
                         rinot         { ring indicator }
                                       : boolean;
                         rdrf          { receive data register full }
                                       : boolean;
                         tdre          { transmit data register empty }
                                       : boolean;
                         dcdnot        { carrier detect not }
                                       : boolean;
                         ctsnot        { cts not }
                                       : boolean;
                         fe            { framing error }
                                       : boolean;
                         ovrn          { receiver overrun }
                                       : boolean;
                         pe            { parity error }
                                       : boolean;
                         irq           { interrupt request }
                                       : boolean;
                       END;

ctrlword        =      PACKED RECORD
                         brs           { bit rate select: 1 = 300 baud }
                                       : baudrate;
                         txe           { transmitter enable }
                                       : boolean;
                         modes         { mode select: 1 = originate }
                                       : mode;
                         cts           { controls cts }
                                       : boolean;
                         st            { self test }
                                       : boolean;
                         filler1       : 0..3;
                         oh            { off hook }
                                       : boolean;
                         cds           { counter divide select }
                                       : 0..3;
                         ws            { word select }
```

```
                                        : 0..7;
                          tc            { transmit control }
                                        : 0..3;
                          rie           { receive interrupt enable }
                                        : boolean;
                        END;


VAR


crt_info        :                       { array of crt command sequences }
                        ARRAY[ crt_command ] OF crt_rec;
ctrl            :       ctrlword;
modem           :       PACKED RECORD
                          CASE integer OF
                            0: ( address1: integer;
                                 address2: integer );
                            1: ( ctrl: ^ ctrlword;
                                 dataout: ^ word );
                            2: ( status: ^ statusword;
                                 datain: ^ word )
                        END;


PROCEDURE show_error( err: error_type );
{ report error }

{-----------------------------------------------}
{  This routine expects a textfile with 88      }
{  lines, each with 23 characters, the first    }
{  two chars are the dle and # of spaces to pad.}
{  ( in this case, ascii 0 ), and the last space}
{  is a carriage return, ascii 13.              }
{  If the file is not there, only the error     }
{  number is reported.                          }
{-----------------------------------------------}

CONST
  error_name    =       '%ERRORS.TEXT';

VAR
  buf           :       page_ptr;
  heap          :       stack_ptr;
  i             :       integer;
  s             :       string;

BEGIN
  gotoxy( 0, error_line );
  clreoln;
  prompt( 'Error ' );
  write( err:2, ': ' );
  {$I-}
  reset( f, error_name );
  {$I+}
  IF ioresult = 0 THEN
    IF min_mem < memavail THEN
    BEGIN
      mark( heap );
      new( buf );
      fillchar( buf^, sizeof( ucsd_page ), 0 );
      i := blockread( f, buf^, 2, ( ( err DIV 44 ) + 1 ) * 2 );
      {$r-}
      s[ 0 ] := chr( 20 );
      {$r+}
      moveleft( buf^[ ( ( err MOD 44 ) * 23 ) + 3 ], s[ 1 ], 20 );
      promptln( s );
      release( heap );
    END;        { if ioresult... }
  close( f );
END;    { show_error }


FUNCTION get_labels(  volume: integer; VAR labels: label_array )
                                                     : error_type };

VAR
  p: path_name;
  iocode: integer;
  pg: page_ptr;
  heap: stack_ptr;
  i: integer;

BEGIN
  str( volume, p );
  p := concat( '#', p, lab_extension );
  {$I-}
  reset( f, p );
  {$I+}
  iocode := ioresult;
  IF iocode = 0 THEN
    BEGIN
      IF min_mem < memavail THEN
        BEGIN
          mark( heap );
          new( pg );
          fillchar( pg^, sizeof( ucsd_page ), 0 );
          i := blockread( f, pg^, 1 );
          moveleft( pg^, labels, sizeof( labels ) );
          release( heap );
        END     { if min_mem... }
      ELSE
        iocode := 21;   { out of memory }
    END;        { if iocode... }
  get_labels := iocode;
  close( f );
END;    { get_labels }

{-------------------------------}
{ Parameter load and update     }
{ procedures.                   }
{-------------------------------}


FUNCTION load_parms( ( vol, block_num: integer ): error_type };
{ load this set of parameters into the correct record }

VAR
  iocode: error_type;
  pg: page_ptr;
  heap: stack_ptr;
  i: integer;
  s: string;

BEGIN
  str( vol, s );
  s := concat( '#', s, prm_extension );
  {$I-}
  reset( f, s );
  {$I+}
  iocode := ioresult;
  IF iocode = 0 THEN
    BEGIN
      IF min_mem < memavail THEN
        BEGIN
          mark( heap );
```

```
     new( pg );
     fillchar( pg^, sizeof( ucsd_page ), 0 );
     i := blockread( f, pg^, 1, block_num );
     iocode := ioresult;
     IF ( i = 1 ) AND ( iocode = 0 )THEN
      BEGIN
       IF block_num IN [ 1..maxposition ] THEN
        moveleft( pg^, cart, sizeof( cart ) )
       ELSE IF block_num = mirec THEN
        moveleft( pg^, misc, sizeof( misc ) )
       ELSE IF block_num = curec THEN
        moveleft( pg^, currvals, sizeof( currvals ) );
      END { if i = 1.. }
     ELSE
      IF iocode = 0 THEN
       iocode := 23;       { undefined error }
     release( heap );
    END     { if min_mem... }
   ELSE
    iocode := 21;   { out of memory }
  END;         { if iocode.. }
 load_parms := iocode;
 close( f );
END;    { load_parms }


FUNCTION update_parms( ( vol, block_num: integer ): error_type );

VAR
 i: integer;
 iocode: error_type;
 heap: stack_ptr;
 pg: page_ptr;
 s: string;

BEGIN
 str( vol, s );
 s := concat( '#', s, prm_extension );
 {$I-}
 reset( f, s );
 {$I+}
 iocode := ioresult;
 IF iocode = 0 THEN
  BEGIN
   IF min_mem < memavail THEN
    BEGIN
     mark( heap );
     new( pg );
     fillchar( pg^, sizeof( ucsd_page ), 0 );
     IF block_num IN [ 1..maxposition ] THEN
      moveleft( cart, pg^, sizeof( cart ) )
     ELSE IF block_num = mirec THEN
      moveleft( misc, pg^, sizeof( misc ) )
     ELSE IF block_num = curec THEN
      moveleft( currvals, pg^, sizeof( currvals ) );
     i := blockwrite( f, pg^, 1, block_num );
     iocode := ioresult;
     IF ( i <> 1 ) AND ( iocode = 0 ) THEN
      iocode := 23;       { undefined error }
     IF iocode = 0 THEN
      close( f, lock );
     release( heap );
    END     { if min_mem... }
   ELSE
    iocode := 21;   { out of memory }
  END;        { if iocode... }
 IF iocode <> 0 THEN
  close( f );     { no update }
  update_parms := iocode;
END;    { update_parms }


{------------------------------}
{  Screen output routines.     }
{  Taken from PDBS.            }
{------------------------------}


PROCEDURE send_crt( c : crt_command );
{ Perform a crt command using crt_info variables }
{ Local to unit }

BEGIN
  WITH crt_info[ c ] DO
    BEGIN
      IF modem_in_use THEN
        unitwrite( 8, crt_ch, crt_n );
      unitwrite( 2, crt_ch, crt_n )
    END;
END; { send_crt }


PROCEDURE evon;
{ Switch on enhanced video }

BEGIN
  send_crt( crt_evon )
END; { evon }


PROCEDURE evoff;
{ Switch off enhanced video }

BEGIN
  send_crt( crt_evoff )
END; { evoff }


PROCEDURE goright;
{ Move cursor non-destructively right }

BEGIN
  send_crt( crt_goright )
END; { goright }


PROCEDURE goup;
{ Move cursor non-destructively up }

BEGIN
  send_crt( crt_goup )
END; { goup }


PROCEDURE beep;
{ Sound the alarm! }

BEGIN
  send_crt( crt_bell )
END; { beep }


PROCEDURE clreoln;
{ Clear from cursor to end of line }

BEGIN
```

```
  send_crt( crt_cleoln )
END; { clreoln }


PROCEDURE clreosc;
{ Clear from cursor to end of screen }

BEGIN
  send_crt( crt_cleosc )
END; { clreosc }


PROCEDURE clrscrn;
{ Clear screen }

BEGIN
  send_crt( crt_clscrn )
END; { clrscrn }


PROCEDURE goxy( x, y: integer );
{ screen movement procedure }

CONST
  offset          =        32;

VAR
  send            :        PACKED ARRAY[ 0..2 ] OF 0..255;

BEGIN
  { The following almost works, but no cigar }
  (* x := abs( x mod 80 ) + 1;     y := abs( y mod 24 );
  send[ 0 ] := 30;                                      { lead-in           }
  send[ 1 ] := x + offset;
  send[ 2 ] := y + offset;
  IF modem_in_use THEN
    unitwrite( 8, send, sizeof( send ) );
  unitwrite( 2, send, sizeof( send ) ); *)
  gotoxy( x, y );
END;    { goxy }


PROCEDURE prompt( s : string );
{ Write string at present cursor position }

BEGIN
  IF modem_in_use THEN
    write( outfile, s );
  write( s )
END; { prompt }


PROCEDURE promptln( s : string );
{ Writeln string at present cursor position }

BEGIN
  IF modem_in_use THEN
    writeln( outfile, s );
  writeln( s )
END; { promptln }


PROCEDURE promptat( x, y : integer; s : string );
{ Write string @ X,Y }

BEGIN
  goxy( x, y );
  prompt( s )
```

```
END; { promptat }


PROCEDURE prompt_real( ( r: real; width, decimals: integer ) );
{ write a real number with specified format }

BEGIN
  IF modem_in_use THEN
    IF decimals = 0 THEN
      write( outfile, trunc( r ):width )
    ELSE
      write( outfile, r:width:decimals );
  IF decimals = 0 THEN
    write( trunc( r ):width )
  ELSE
    write( r:width:decimals );
END;    { prompt_real }


{------------------------------------------------------------}
{ disp_screen                                                 }
{                                                             }
{ Read and display a menu.  Menu is stored in the form of     }
{ a Pascal textfile which can be edited using the Pascal      }
{ text editor.  An 80 column by 24 line display is supported. }
{ Each menu is named with the keyword "SCREEN" followed by    }
{ a menu number.  All screens are stored on a separate        }
{ volume named SCREENS:.                                      }
{------------------------------------------------------------}


PROCEDURE disp_screen( ( screen_num: integer ) );

VAR
  buf            : page_ptr;
  s              : string;
  io_code,
  bytes_scanned,
  page,
  blocks         : integer;
  heap           : stack_ptr;
  ch             : char;


PROCEDURE getbuffer( VAR buf: ucsd_page; relblock: integer;
                                         VAR blkcount: integer);
{ read menu into buffer }

BEGIN
  fillchar( buf, sizeof( buf ), 0 );
  blkcount := blockread( f, buf, 2, relblock );
  IF blkcount = 2 THEN
    bytes_scanned := scan( -1024, <> chr( 0 ), buf[ 1024 ] )
  ELSE
    bytes_scanned := 0;
END;    { getbuffer }


PROCEDURE dooutput;
{ display menu }

BEGIN
  FOR page := 1 TO 2 DO
    BEGIN
      getbuffer( buf^, ( page * 2 ), blocks );
      IF blocks = 2 THEN
        BEGIN
          IF modem_in_use THEN
```

```
            BEGIN
              unitwrite( 8, buf^, sizeof( ucsd_page ) + bytes_scanned, 0, 0 );
              stopwatch( 0.25 );
            END;
          unitwrite( 2, buf^, sizeof( ucsd_page ) + bytes_scanned, 0, 0 );
        END;    { if blocks... }
    END;
END;   { dooutput }


BEGIN   { disp_screen }
  IF min_mem < memavail THEN
    BEGIN
      mark( heap );
      new( buf );
      str( screen_num, s );
      s := concat( 'SCREENS:SCREEN', s, '.TEXT' );
      {$I-}
      reset( f, s );
      {$I+}
      io_code := ioresult;
      IF io_code = 0 THEN
        dooutput
      ELSE
        show_error( io_code );
      close( f );
      release( heap );
    END
  ELSE
    show_error( 21 );   { out of memory }
END;  { disp_screen }


{---------------------------------------}
{ keyboard input routines               }
{---------------------------------------}


FUNCTION keypress{: boolean };
  external;


FUNCTION getchar{ okset: setofchar; echo, sensitive: boolean ): char };
{ gets character in okset, all others beeped.              }
{ if not sensitive, then shift the character to upper case }

VAR
  ch            : char;
  good          : boolean;

BEGIN
  REPEAT
    read( keyboard, ch );
    IF NOT sensitive THEN
      upcase( ch );
    IF eoln( keyboard ) THEN ch := ret;
    good := ch IN okset;
    IF NOT good THEN beep
    ELSE IF ( ( ch IN [ space..chr( 125 ) ] ) AND echo )
      THEN write( ch );
  UNTIL good;
  getchar := ch;
END;  { getchar }


FUNCTION finished; { ( firstchar: boolean;        first char in entry?     }
                   {   ch: char;                  char to test             }
                   {   end_char_set: setofchar ) : boolean                 }
{ used by getstring and getnum. determine when entry is finished.         }
{ end_char_set is an extra set of ending characters. i.e. space for numbers.}
{ cmd_set is the set of commands which are applicable only at the beginning }
{ of the entry.  set up cmd_set in the calling program.                   }

BEGIN
  finished := ( firstchar AND ( ( ch IN cmd_set + [ keyleft ] ) ) ) OR
              ( ch IN ( exit_set + end_char_set ) );
END;   { finished }


FUNCTION fpnum((fpstr: string):real);

VAR power, sign, i       :integer;
    num                  :real;

BEGIN
  IF fpstr[ 1 ] = minus THEN
    BEGIN
      sign := -1;
      delete( fpstr, 1, 1);
    END
  ELSE sign := 1;
  power := pos( '.', fpstr );
  IF power <> 0 THEN
    BEGIN
      delete( fpstr, power, 1 );
      power := length( fpstr ) - power + 1;
    END;
  num := 0;
  FOR i := 1 TO length( fpstr ) DO
    num := 10 * num + ( ord( fpstr[ i ] ) - ord( '0' ) );
  fpnum := sign * num / pwroften( power );
END;  { fpnum }


FUNCTION getnum( ( oldval, maxval, minval: real; ptok: boolean;
                                               VAR last_char: char ): real );
{ used by getfpnum and getinteger }

VAR     done, firstchar, endchar         :boolean;
        numset,  getset,  okset          :setofchar;
        numstrtemp                       :string[10];
        s1                               :char;
        len, maxlen, i                   :integer;
        num                              :real;

BEGIN
  numset := [ '0'..'9' ];
  IF minval < 0 THEN
    okset := numset + [ minus ] ELSE okset := numset;
  IF ptok THEN maxlen := 8 ELSE maxlen := 7;
  s1 := space;  numstrtemp := '';  last_char := space;
  done := false;

  REPEAT
    len := length( numstrtemp );
    firstchar := ( len = 0 );
    endchar := ( len = maxlen );

    IF ptok THEN okset := okset + [ '.' ]
    ELSE okset := okset - [ '.' ];

    IF firstchar THEN
      getset := okset + exit_set + [ space ] + cmd_set + [ keyleft ]
  ELSE
      IF NOT endchar THEN
```

```
        getset := okset + exit_set + [ space ] - [ minus ] + [ keyleft ]
      ELSE getset := exit_set + [ space ] + [ keyleft ];

    s1 := getchar( getset, true, false );
    IF s1 = '.' THEN ptok := false;

    IF s1 = esc THEN
      BEGIN
        numstrtemp := '';
        len := 0;
      END;

    done := finished( firstchar, s1, [ space ] );

    IF done THEN
      BEGIN
        upcase( s1 );
        last_char := s1;
      END;      { if done }

    IF ( s1 IN okset ) AND ( NOT done ) THEN
      BEGIN
        numstrtemp := concat( numstrtemp, ' ' );
        numstrtemp[ length( numstrtemp ) ] := s1;
      END

    ELSE IF ( s1 = keyleft ) AND ( NOT done ) THEN
      BEGIN
        IF pos( '.', numstrtemp ) = len THEN
          ptok:=true;
        write( keyleft, space, keyleft );
        delete( numstrtemp, len, 1 );
      END;

    IF done AND ( len > 0 ) THEN
      BEGIN
        num := fpnum( numstrtemp );
        IF ( num > maxval ) OR ( num < minval ) THEN
          BEGIN
            IF pos( '.', numstrtemp ) > 0 THEN ptok := true;
            beep;
            done := false;
            numstrtemp := '';
            s1 := space;
            FOR i := len DOWNTO 1 DO
              write( keyleft, space, keyleft);
          END;  { if ( num > maxval )  etc..... }
      END;  { if done ... }

  UNTIL done;  writeln;

  IF length( numstrtemp ) > 0 THEN
    getnum := fpnum( numstrtemp )
  ELSE getnum := oldval;

END;  { getnum }


FUNCTION getinteger( ( default, maxval, minval: integer;var lastchar: char
                                                                ): integer );
{ gets integer number between specified max and min values }

BEGIN
  getinteger := trunc( getnum( default, maxval, minval, false, last_char ) );
END;  { getinteger }


{------------------------------}
```

```
{ Various goodies                 }
{-------------------------------}


FUNCTION read_time(: integer );
  external;


FUNCTION verified( s : string ) : boolean );
{ True if 'Y' or 'y' typed in response to s }

VAR
  ch : char;

BEGIN
  prompt( concat( s, ' ? (y/n) ') );
  ch := getchar( [ 'Y', 'N' ], true, false );
  verified := ( ch IN [ 'Y' ] );
END;  { verified }


FUNCTION executed( e: error_type ) : boolean );
{ Execute any function that returns a pdbs error code. }
{ Returns true if no error, else prints message and returns false. }

BEGIN { executed }
  IF e <> 0 THEN
    show_error( e );
  executed := ( e = 0 )
END; { executed }


FUNCTION elapsedtime( ( nowdate: daterec; nowtime, nowsecs: integer;
                        thendate: daterec; thentime, thensecs: integer ): real );
{ returns the elapsed time between dates in hours and fractions of hours }

VAR
  nowjd,
  thenjd,
  nowhours,
  nowmin,
  thenhours,
  thenmin               :       integer;

FUNCTION julian( day, month: integer ): integer;
{ returns the julian day of a non-leap year }

VAR
  i,
  totdays               :       integer;

FUNCTION daysinmo( month: integer ): integer;

BEGIN
  CASE month OF
    1,3,5,7,8,10,12 : daysinmo := 31;
    2               : daysinmo := 28;
    4,6,9,11        : daysinmo := 30;
  END;  { case }
END;   { daysinmo }

BEGIN   { julian }
  totdays := 0;
  FOR i := 1 TO month - 1 DO
    totdays := totdays + daysinmo(i);
  julian := totdays + day;
END;  { julian }
```

```
BEGIN  { elapsed time }
  nowhours := nowtime DIV 100;
  nowmin := nowtime MOD 100;
  thenhours := thentime DIV 100;
  thenmin := thentime MOD 100;
  nowjd := julian( nowdate.day, nowdate.mo );
  thenjd := julian( thendate.day, thendate.mo );
  elapsedtime := ( ( nowdate.yr - thendate.yr ) * 8760 ) +
                 ( ( nowjd - thenjd ) * 24.0 ) +
                 ( ( nowhours - thenhours) + ( ( nowmin - thenmin ) / 60 ) ) +
                 ( ( nowsecs - thensecs ) / 3600 );
END;  { elapsedtime }


FUNCTION calcsecs({lasttime,lastsecs: integer):integer};
{ returns difference between current time and lasttime in seconds }
{ returns at most 32400 seconds }

VAR
  temp: real;

BEGIN
  temp := elapsedtime( date, time, seconds, date, lasttime, lastsecs );
  WHILE temp < 0 DO
    temp := temp + 24;
  IF temp > 9 THEN temp := 9;
  calcsecs := trunc( temp * 3600 );
END;   { calcsecs }


PROCEDURE perform( e: error_type );
{ Execute any function that returns a pdbs error code. }
{ Displays error if there is a problem. }

BEGIN { perform }
  IF e <> 0 THEN
    show_error( e );
END; { perform }


PROCEDURE wait( time: integer );
{ wait "time" milliseconds }

VAR
  i,
  dummy: integer;

BEGIN
  FOR i := 1 TO time DO
    BEGIN
      dummy := 0;
      dummy := 1;
      dummy := 2;
    END;
END;  { wait }


PROCEDURE stopwatch{ ( seconds: real ) };
{ exact time delay routine }

BEGIN   { stopwatch, uses the wait procedure in the micromodem unit }
  WHILE seconds > 30 DO
    BEGIN
      seconds := seconds - 30;
      wait( 30000 );
    END;
  wait( round( seconds * 1000 ) );
END;  {stopwatch}
```

```
PROCEDURE date_2_string( date : ucsd_date; VAR date_string : pdbs_string );
{ Convert UCSD system date into a string s }

VAR
  day_string,
  year_string : string;

PROCEDURE num2string( n, d: integer; pad: char; VAR s: string );
{ convert number n to string s padded at left to d digits with pad }

VAR
  s1: string[ 1 ];

BEGIN
  s1 := ' ';
  s1[ 1 ] := pad;
  str( n, s );
  WHILE length( s ) < d DO
    insert( s1, s, 1 );
END;    { num2string }

BEGIN   { date_2_string }
  WITH date DO
    BEGIN
      num2string( day, 2, '0', day_string );
      num2string( yr, 2, '0', year_string );
      date_string := concat( day_string, '-',
                             copy( month_names, ( mo * 3 + 1 ), 3 ),
                             '-', year_string )
    END { with }
END; { date_2_string }


PROCEDURE upcase( ch: char );
{ put ch into upper case }

BEGIN
  IF ch IN [ 'a'..'z' ] THEN ch := chr( ord( ch ) - 32 );
END;    { upcase }


{-------------------------------}
{  Modem procedures             }
{-------------------------------}


FUNCTION ringing;
{ test ring indication }

BEGIN
  ringing := NOT modem.status^.rinot;
END;  { ringing }


FUNCTION carrier;
{ test for presence of carrier }

BEGIN
  carrier := NOT modem.status^.dcdnot;
END;    { carrier }


FUNCTION rcvrfull;
{ check for a character in the input buffer }

BEGIN
```

```
  rcvrfull := modem.status^.rdrf;
END;    { rcvrfull }


PROCEDURE initcomm; { wordselect: integer }
{ reset and initialize acia }

VAR
  wait: integer;

BEGIN
  ctrl.cds := 3;  { master reset }
  modem.ctrl^ := ctrl;
  wait := 0;
  REPEAT
    wait := succ(wait)
  UNTIL NOT( carrier ) OR ( wait > 5000 );
  WITH ctrl DO
    BEGIN
      brs := high;
      txe := false;
      modes := originate;
      cts := true;
      st := false;
      filler1 := 0;
      oh := false;
      cds := 1;
      ws := wordselect MOD 8;
      tc := 0;
      rie := false;
    END;
  modem.ctrl^ := ctrl;
END;   { initcomm }


PROCEDURE setmode; { md: mode }
{ set the mode }

BEGIN
  ctrl.modes := md;
  modem.ctrl^ := ctrl;
END;  { setmode }


PROCEDURE enabletransmit;
{ turn on the modem transmitter }

BEGIN
  ctrl.txe := true;
  modem.ctrl^ := ctrl;
END;  { enabletransmit }


PROCEDURE disabletransmit;
{ turn off the modem transmitter }

BEGIN
  ctrl.txe := false;
  modem.ctrl^ := ctrl;
END;  { disabletransmit }


PROCEDURE pickup;
{ place phone off-hook }

BEGIN
  ctrl.oh := true;
  modem.ctrl^ := ctrl;
```

```
END;  { pickup }


PROCEDURE hangup;
{ place phone on-hook }

BEGIN
  ctrl.oh := false;
  modem.ctrl^ := ctrl;
END;  { hangup }


PROCEDURE unconnect;
{ break connection }

VAR
  ch: char;
  wait: integer;

BEGIN
  disabletransmit;
  hangup;
  wait := 0;
  WHILE carrier AND (wait < 5000) DO
    BEGIN
      wait := succ(wait);
      ch := modem.datain^[0];
    END;
END;  { unconnect }


PROCEDURE dial; { number: string }
{ dial the indicated number }

VAR
  ch: char;
  i,
  j,
  digit: integer;

BEGIN
  IF NOT( ctrl.oh ) AND ( length( number ) <> 0 ) THEN
    BEGIN
      gotoxy( 0, error_line );
      clreoln;
      evon;
      prompt('<ctrl-c> aborts auto dial --> ');
      evoff;
      setmode(originate);
      pickup;
      wait( 1400 );
      prompt(' Ok, dialing...');
      FOR i := 1 TO length(number) DO
        BEGIN
          IF keypress THEN
            BEGIN
              ch := getchar( [ chr( 0 )..chr( 127 ) ], false, false );
              if ch = esc then
                begin
                  prompt( ' >>> aborted <<<' );
                  unconnect;
                  exit(dial);
                end;    { if ch = esc }
            END;        { if keypress }
          IF number[ i ] IN [ '0'..'9' ] THEN
            BEGIN
              digit := ord( number[ i ] ) - ord( '0' );
              prompt_real( digit, 1, 0 );
              IF digit = 0 THEN
                digit := 10;
              wait( 600 );
              FOR j := 1 TO digit DO
                BEGIN
                  hangup;
                  wait( 61 );
                  pickup;
                  wait( 39 );
                END;
            END
          ELSE IF number[ i ] = '*' THEN
            begin
              prompt( '*' );
              wait( 2000 );
            end;
        END;
      { waitforcarrier }
    END;
END;  { dial }


PROCEDURE answerthephone;

VAR
  i     :       integer;
  ch    :       char;

BEGIN
  setmode(answer);
  pickup;
  enabletransmit;
  i := 0;
  REPEAT
    i := i + 1;
    ch := modem.datain^[ 0 ];
  UNTIL carrier OR ( i > 4500 );
END;    { answerthephone }


PROCEDURE initialize;
{ unit initialization - mostly taken from PDBS }

VAR
  buf   :       page_ptr;
  heap  :       stack_ptr;
  misc_entered: boolean;

PROCEDURE read_crt;
{ Read crt (display) commands from PDBS.MISCINFO file }

CONST
  crt_fname = '%miscinfo';

BEGIN
   mark( heap );
   new( buf );
   fillchar( buf^, sizeof( ucsd_page ), 0 );
   misc_entered := true;
   {$I-}
   reset( f, crt_fname );
   {$I+}
   IF ioresult <> 0 THEN
     misc_entered := false
   ELSE IF blockread( f, buf^, 1, 0 ) <> 1 THEN
      misc_entered := false;
   IF NOT misc_entered THEN
     BEGIN
```

```
       writeln( 'Can''t read ', crt_fname );
       halt
     END;
   moveleft( buf^, crt_info, sizeof( crt_info ) );
   close( f );
   release( heap );
END; { read_crt }

BEGIN   { initialize }
  page( output );
  writeln;
  writeln( 'Unit Initialization... ' );
  time := readtime;                       { sets date, time and seconds }
  read_crt;                               { Read %MISCINFO crt commands }
  ret := chr( 13 );      plus := chr( 43 );      minus := chr( 45 );
  keyleft := chr( 8 );    keyright := chr( 21 );     keyup := chr( 11 );
  keydown := chr( 10 );   esc := chr( 27 );       space := chr( 32 );
  keyetx := chr( 3 );
  mark_char[ marked ] := '*';  mark_char[ unmarked ] := '+';
  fillchar( labels, sizeof( labels ), 0 );
  fillchar( keys, sizeof( keys ), 0 );
  cmd_set := [ ];
  exit_set := [ ret, keyright, esc, keyup, keydown, keyetx ];
  modem_in_use := false;
  modem.address1 := -16219;
  modem.address2 := -16217;
  goxy( 0, 0 );
  clrscrn;
END;    { initialize }
```

```
{ L wrk.lst.text}
UNIT work_stuff;

{-----------------------------------------------}
{  Work_stuff unit - this is the unit which is  }
{  required for taking measurements and working }
{  with the statpro workfiles which are         }
{  associated with the Control program.         }
{  Contributers are:                            }
{  Allen Todd -         Assembly procedures     }
{                                               }
{  Version 10 Oct 85    1300    -mkm-           }
{-----------------------------------------------}


INTERFACE

USES
  {$U ctrlstuff.code}  control_stuff;


VAR
  workfile      :       FILE OF workrec;
  wrk           :       workrec;
  switchstate   :       PACKED ARRAY[ 1..maxswitch ] OF boolean;
  interupt      :       boolean;


{-------------------------------}
{  Workfile procedures          }
{-------------------------------}

FUNCTION open_workfile( volume: integer ): error_type;
FUNCTION get_keys( volume: integer; VAR keys: keyrec ): error_type;
FUNCTION get_rec( recnum: integer; VAR w: workrec ): error_type;
FUNCTION append_rec( VAR recnum: integer; w: workrec ): error_type;
FUNCTION update_rec( recnum: integer; w: workrec ): error_type;
FUNCTION update_keys( volume: integer; keys: keyrec ): error_type;

PROCEDURE zero_wrk( VAR w: workrec );
PROCEDURE close_workfile;

{-------------------------------}
{ Date, time conversions        }
{-------------------------------}

FUNCTION date_2_r( date: ucsd_date ): real;
FUNCTION time_2_r( time, seconds: integer ): real;

PROCEDURE r_2_date( r: real; VAR date: ucsd_date );
PROCEDURE r_2_time( r: real; VAR time, seconds: integer );

{-------------------------------}
{ Assembly procedures           }
{-------------------------------}

FUNCTION analog( channel: integer; range: voltrange ): integer;

PROCEDURE switch( channel: integer; state: onoff; update: boolean );
PROCEDURE cart_select( cartnum: integer );
PROCEDURE gas_select( valve: integer );
PROCEDURE dac ( value: integer );

{-------------------------------}
{ conversion routines           }
{-------------------------------}

FUNCTION ad_2_volts( value: integer; range: voltrange ): real;
```

```
PROCEDURE volts_2_scientific( field: integer;   { the field # of the data   }
                              eqns: equations;  { conversion equation       }
                              VAR data: real ); { original->new data value  }

{-------------------------------}
{ Channel measurement function  }
{-------------------------------}

FUNCTION integrate_a_2_d( chan: integer; vr: voltrange ): real;

{-------------------------------}
{ Rate calculation function     }
{-------------------------------}

FUNCTION slope( sx, sxsqr, sxy, sy, n: real ): real;

{-------------------------------}
{ Max min calculations          }
{-------------------------------}

FUNCTION max( a, b: real ): real;
FUNCTION min( a, b: real ): real;


IMPLEMENTATION


{-------------------------------}
{  workfile routines            }
{-------------------------------}

{$I-}

FUNCTION open_workfile( ( volume: integer ): error_type );
{ Tries to reset a workfile, returns ioresult }

VAR
  p: pathname;

BEGIN
  str( volume, p );
  p := concat( '#', p, wrk_extension );
  reset( workfile, p );
  open_workfile := ioresult;
END;    { open_workfile }


FUNCTION get_keys( ( volume: integer; var keys: keyrec ): error_type );
{ Tries to open a keyfile.  If successful, loads keys record }

VAR
  p: pathname;
  iocode: error_type;
  pg: page_ptr;
  heap: stack_ptr;
  i: integer;

BEGIN
  str( volume, p );
  p := concat( '#', p, key_extension );
  reset( f, p );
  iocode := ioresult;
  IF iocode = 0 THEN
    BEGIN
      IF min_mem < memavail THEN
        BEGIN
          mark( heap );
          new( pg );
          fillchar( pg^, sizeof( ucsd_page ), 0 );
          i := blockread( f, pg^, 1 );
          moveleft( pg^, keys, sizeof( keys ) );
          release( heap );
        END          { if min_mem... }
      ELSE
        iocode := 21;   { out of memory }
    END;        { if iocode... }
  get_keys := iocode;
  close( f );
END;    { get_keys }


FUNCTION get_rec{ ( recnum: integer; var w: workrec ): error_type };
{ puts the appropriate record into w.  Note that workfile^ can also }
{ be loaded the same way, i.e. the program can access workfile^ as  }
{ well as w. }

VAR
  iocode        :       integer;

BEGIN
  seek( workfile, recnum );
  get( workfile );
  iocode := ioresult;
  IF iocode = 0 THEN
    w := workfile^;
  get_rec := iocode;
END;    { get_rec }


FUNCTION append_rec{ ( var recnum: integer; w: workrec ): error_type };
{ Adds w to the record specified and adds one to recnum }
{ can be used for replacing records in the middle of file if necessary }
{ if unsuccessful, recnum remains the same }

VAR
  iocode: error_type;

BEGIN
  seek( workfile, recnum );
  workfile^ := w;
  put( workfile );
  iocode := ioresult;
  IF iocode = 0 THEN
    recnum := recnum + 1;
  append_rec := iocode;
END;    { append_rec }


FUNCTION update_rec{ ( recnum: integer; w: workrec ): error_type };
{ same as append_rec, but recnum not updated }

BEGIN
  seek( workfile, recnum );
  workfile^ := w;
  put( workfile );
  update_rec := ioresult;
END;    { update_rec }


FUNCTION update_keys{ ( volume: integer; keys: keyrec ): error_type };

VAR
  i: integer;
  p: pathname;
  iocode: error_type;
  heap: stack_ptr;
  pg: page_ptr;

BEGIN
  str( volume, p );
  p := concat( '#', p, key_extension );
  reset( f, p );
  iocode := ioresult;
  IF iocode = 0 THEN
    BEGIN
      IF min_mem < memavail THEN
        BEGIN
          mark( heap );
          new( pg );
          fillchar( pg^, sizeof( ucsd_page ), 0 );
          moveleft( keys, pg^, sizeof( keys ) );
          i := blockwrite( f, pg^, 1 );
          IF i < 1 THEN
            iocode := 8     { No room on disk error }
          ELSE
            close( f, lock );
          release( heap );
        END
      ELSE
        iocode := 21; { memory error }
    END;         { if iocode... }
  IF iocode <> 0 THEN
    close( f ); { no update }
  update_keys := ioresult;
END;    { update_keys }


PROCEDURE zero_wrk( var w: workrec );

VAR
  i     :       integer;

BEGIN
  FOR i := 1 TO max_field DO
    w.data[ i ] := missing;
  w.dflag := mark_char[ unmarked ];
END;    { zero_wrk }


PROCEDURE close_workfile;

BEGIN
  close( workfile, lock );
END;    { close_workfile }

{$I+}

FUNCTION date_2_r{ ( date: ucsd_date ): real };
{ convert the date to MM.DDYY real format                            }

BEGIN
  WITH date DO
    date_2_r := mo + ( day / 100 ) + ( yr / 10000 );
END;    { date_2_r }


FUNCTION time_2_r{ ( time, seconds: integer ): real };
{ convert the time to HH.MMSS format                                 }

BEGIN
  time_2_r := ( time DIV 100 ) +             { add the hours         }
              ( ( time MOD 100 ) / 100 ) +   { the minutes           }
              ( seconds / 10000 );           { and the seconds       }
END;    { time_2_r }
```

```
PROCEDURE r_2_date{ ( r: real; VAR date: ucsd_date ) };
{ Convert MM.DDYY format real number to ucsd date format.                }

VAR
  m,
  d,
  y     :       integer;

BEGIN
  m := trunc( r );
  y := round( ( ( r  *  100 ) - trunc( r * 100 ) ) * 100 );
  d := round( ( ( trunc( r * 100 ) / 100 ) -
            ( trunc( trunc( r * 100 ) / 100 ) ) ) * 100 );
  WITH date DO
    IF ( m IN [ 1..12 ] ) AND ( y IN [ 1..100 ] ) AND ( d IN [ 1..31 ] ) THEN
      BEGIN
        mo := m;         day := d;        yr := y;
      END
    ELSE
      BEGIN
        mo := 0;         day := 0;        yr := 0;
      END;
END;    { r_2_date }


PROCEDURE r_2_time{ ( r: real; VAR time, seconds: integer ) };
{ Convert HH.MMSS real number into the Control time representation HHMM, SS }


VAR
  h,
  m,
  s     :       integer;

BEGIN
  h := trunc( r );
  s := round( ( ( r * 100 ) - trunc( r * 100 ) ) * 100 );
  m := round( ( ( trunc( r * 100 ) / 100 ) -
            ( trunc( trunc( r * 100 ) / 100 ) ) ) * 100 );
  IF ( h IN [ 0..23 ] ) AND ( m IN [ 0..59 ] ) AND ( s IN [ 0..59 ] ) THEN
    BEGIN
      time := trunc( h * 100 + m );
      seconds := s;
    END
  ELSE
    BEGIN
      time := 0;
      seconds := 0;
    END;
END;    { r_2_time }


{-------------------------------}
{ Assembly procedures           }
{-------------------------------}

FUNCTION do_analog( channel: integer; range: voltrange ): integer;
  external;

FUNCTION analog{ ( channel: integer; range: voltrange ): integer };

BEGIN   { analog }
  analog := do_analog( channel - 1, range );
END;    { analog }
```

```
  PROCEDURE do_switch( channel: integer: state: onoff );
    external;

  PROCEDURE switch( channel: integer; state: onoff; update: boolean );

  BEGIN   { switch }
    do_switch( channel - 1, state );
    IF state = on THEN
      currvals.switchvals[ channel ] := true
    ELSE
      currvals.switchvals[ channel ] := false;
    IF update THEN
      perform( update_parms( currvals.workvolume, curec ) );
  END;    { switch }


  PROCEDURE initdrivers;
    external;


  PROCEDURE do_cart_select( cartnum: integer );
    external;

  PROCEDURE cart_select( cartnum: integer );

  BEGIN   { cart_select }
    do_cart_select( cartnum - 1 );
  END;    { cart_select }


  PROCEDURE do_gas_select( valve: integer );
    external;

  PROCEDURE gas_select( valve: integer );

  BEGIN   { gas_select }
    do_gas_select( valve - 1 );
  END;    { gas_select }


  PROCEDURE dac ( value: integer );
    external;


  FUNCTION ad_2_volts( ( value: integer; range: voltrange ): real );

  VAR
    x,
    offset          :       real;

  BEGIN
    offset := 0;
    CASE range OF
      plus5v,
      dual5v,
      disconnect  :       x := 5.0;
      plus1v,
      dual1v      :       x:= 1.0;
      plus500mv,
      dual500mv   :       x:= 0.5;
      plus100mv,
      dual100mv   :       x:= 0.1;
    END;
    IF range IN [ dual5v, dual1v, dual500mv, dual100mv ] THEN
      ad_2_volts := ( ( value * x * 4.884E-4 ) - x )
    ELSE
      ad_2_volts := ( value * x * 2.442E-4 );
END;  { ad_2_volts }
```

```
PROCEDURE volts_2_scientific( field: integer;
                              eqns: equations;
                              VAR data: real );
{ converts voltage reading into scientific value by using the conversion }
{ equations.                                                             }

VAR
  volts,
  value :       real;
  i     :       integer;

BEGIN
  volts := data;
  IF volts <> missing THEN
    IF volts <> 0 THEN
      BEGIN
        value := eqns[ 1, field ];
        FOR i := 2 TO 5 DO
          BEGIN
            value := value + eqns[ i, field ] * volts;
            volts := data * volts;
          END;
        data := value;
      END
    ELSE
      data := eqns[ 1, field ]
  ELSE
    data := missing;
END;    { volts_2_scientific }


FUNCTION integrate_a_2_d{ ( chan: integer; vr: voltrange ): real };
{ return the average reading of 20 measurements of this channel             }

VAR
  sum   :       real;
  n     :       integer;

BEGIN
  sum := 0.0;
  FOR n := 1 TO samplecount DO
    sum := sum + a_d_2_volts( analog( chan, vr ), vr );
  integrate_a_2_d := sum / samplecount;
END;    { integrate_a_2_d }

FUNCTION slope{ ( sx, sxsqr, sxy, sy, n: real ): real };
{ Calculate the slope of a line.                                            }
{ Pass the sum of the x's, the sum of the squared x's, the sum of the       }
{ x * y's, the sum of the y's and the number of observations.               }

VAR
  denominator   :       real;

BEGIN
  IF n > 0 THEN
    BEGIN
      { slope := ( n * sxy - sy * sx ) / ( n * sxsqr - sqr( sx ) ); }
      denominator := n * sxsqr - sqr( sx );
      IF denominator <> 0 THEN
        slope := ( n * sxy - sx * sy ) / denominator
      ELSE
        slope := missing;
    END
  ELSE
    slope := missing;
END;    { slope }
```

```
FUNCTION max{ ( a, b: real ): real };

BEGIN
  IF a > b THEN
    max := a
  ELSE
    max := b;
END;    { max }


FUNCTION min{ ( a, b: real ): real };

BEGIN
  IF a < b THEN
    min := a
  ELSE
    min := b;
END;    { min }


BEGIN
  initdrivers;
END.    { workstuff }
```

MAIN PROGRAM

The CONTROL program source consists of four files:

a) CONTROL.TEXT (See page 2-50)
   Contains the highest level code for the program.
b) CONTROL1.TEXT (See page 2-62)
   Contains segment procedures initialize, statdisplay, and airstuff.
c) CONTROL2.TEXT (See page 2-72)
   Contains segment procedures measure and sumstats.
d) CONTROL3.TEXT (See page 2-82)
   Contains segment procedures adjust, quit and call.

```
{ L cont.lst.text}
PROGRAM controller;
{----------------------------------------------}
{ Control Program -                            }
{                                              }
{ Program is designed to monitor and control   }
{ the environments of plant exposure chambers. }
{ Written for the Toxics and Pesticides Branch }
{ plant exposure experiments.                  }
{                                              }
{ By: Mike McDowell                            }
{      EPA/CERL                                }
{      200 SW 35th St.                         }
{      Corvallis, OR   97333                   }
{      (503) 757-4604                          }
{                                              }
{ Contributors:                                }
{ Allen Todd            Assembly, original    .}
{                       program.               }
{ Robert Clark          Telephone alert        }
{ Chris Wilson          Modem routines         }
{ Ron DeGroat           Keyboard input         }
{                                              }
{ Version  11 Oct 85    1600                   }
{----------------------------------------------}

USES
   transcend,
   chainstuff,
   ($U ctrlstuff.code)  control_stuff,
   ($U wrkstuff.code)   work_stuff;

FUNCTION measinterrupt: boolean;
{ checks to see if a regular interrupt should occur }
   forward;

FUNCTION airinterrupt: boolean;
{ checks to see if an air interrupt should occur }
   forward;

PROCEDURE closemodemfiles;
{ hang up phone, stop sending output to outfile }
   forward;

PROCEDURE openmodemfiles;
{ start sending output to REMOUT:, accept input from REMIN: }
   forward;

PROCEDURE writedate;
{ write the date on the screen }
   forward;

PROCEDURE writetime;
{ write the time on the screen }
   forward;

PROCEDURE writemem;
{ write memavail on the screen }
   forward;

PROCEDURE writeair;
{ write current airstring on the screen }
   forward;

PROCEDURE writebools;
{ write system on/off indicators, +/- }
   forward;
```

```
PROCEDURE updatedisplay;
{ call all "write" procedures above to update the screen }
   forward;

PROCEDURE updatecurrvals;
   forward;

PROCEDURE footing;
{ display prompt at bottom of screen }
   forward;


{$I CONTROL1.TEXT}                     { Segment                       }
{$I CONTROL2.TEXT}                     {                               }
{$I CONTROL3.TEXT}                     { procedures                    }


FUNCTION measinterrupt(: boolean);

VAR temp: boolean;

BEGIN
  temp := false;
  IF ( currvals.regmeasure ) THEN
    IF ( ( currvals.aircondition <> gasmeas ) OR
        ( NOT currvals.airmeasure ) ) THEN
      IF calcsecs( samptime, sampseconds ) > misc.samprate THEN
        BEGIN
          samptime := time;
          sampseconds := seconds;
          temp := true;
        END;
  measinterrupt := temp;
END;  { measinterrupt }


FUNCTION airinterrupt(: boolean);

BEGIN   { airinterrupt }
  airinterrupt := false;
  IF currvals.airmeasure THEN
    IF calcsecs( airtime, airseconds ) > misc.airsamprate THEN
      airinterrupt := true;
  { airtime and airseconds are updated in airstuff }
END;  { airinterrupt }


PROCEDURE toggle( VAR totog: boolean; s: string );
{ toggle air or regular interrupts }

VAR ch: char;

BEGIN
  goxy( 0, error_line );
  clreoln;
  prompt( concat( 'Toggle ', s, '? [Y/N] ' ) );
  ch := getchar( ['Y', 'N' ], true, false );
  IF ch = 'Y' THEN
    totog := NOT totog;
  goxy( 0, error_line );
  clreoln;
  footing;
END;  { toggle }


PROCEDURE airtog;

VAR
```

```
    temp: boolean;

  PROCEDURE switchesoff;

  BEGIN
    switch( airpump, off, false );
    switch( airswitch, off, false );
    switch( vent, off, false );
    switch( n2gas, off, false );
    switch( co2stdgas, off, true );
  END;   { switchesoff }

  BEGIN   { airtog }
    temp :=  currvals.airmeasure;
    IF ( NOT currvals.airmeasure ) THEN
      BEGIN
        airtime := time;
        airseconds := seconds;
      END;
    toggle( temp, 'Air interrupts' );
    IF temp <> currvals.airmeasure THEN
      IF ( NOT temp ) THEN
        BEGIN
          currvals.airstring := ( '         Off' );
          switchesoff;
        END;
    currvals.airmeasure := temp;
    perform( update_parms( currvals.workvolume, curec ) );
    writebools;
  END;  { airtog }

  PROCEDURE regtog;

  BEGIN
    toggle( currvals.regmeasure, 'Regular interrupts' );
    IF currvals.regmeasure THEN
      BEGIN
        samptime := time;
        sampseconds := seconds;
      END
    ELSE
      currvals.beep := false;                    { turn off beeping          }
    perform( update_parms( currvals.workvolume, curec ) );
    writebools;
  END;  { regtog }


  PROCEDURE phonetog;
  BEGIN
    toggle( currvals.answerok, 'Phone answering' );
    perform( update_parms( currvals.workvolume, curec ) );
    writebools;
  END;   { phonetog }


  PROCEDURE statstog;

  VAR
    temp : boolean;

  BEGIN
    temp := currvals.stat_collect;
    toggle( currvals.stat_collect, 'Statistics collection' );
    perform( update_parms( currvals.workvolume, curec ) );
    writebools;
  END;    { statstog }
```

```
PROCEDURE datatog;

BEGIN
  toggle( currvals.data_collect, 'Data collection' );
  IF NOT( currvals.data_collect ) THEN
    currvals.beep := false;
  perform( update_parms( currvals.workvolume, curec ) );
  writebools;
END;     { datatog }


PROCEDURE updatecurrvals;

VAR i: integer;

BEGIN
  goxy( 0, error_line );
  clreoln;
  prompt( 'Updating current program values ' );
  FOR i := 1 TO maxswitch DO
    IF switchstate[ i ] THEN
      currvals.switchvals[ i ] := true
    ELSE
      currvals.switchvals[ i ] := false;
  perform( update_parms( currvals.workvolume, curec ) );
  goxy( 0, error_line );
  clreoln;
  footing;
END;   { updatecurrvals }


PROCEDURE writeair;

BEGIN
  goxy( 12, 1 );
  prompt( currvals.airstring );
END;  { writeair }


PROCEDURE writebools;

PROCEDURE printplusorminus( bool: boolean );

BEGIN
  IF bool THEN
    prompt( '+' )
  ELSE
    prompt( '-' );
END;    { printplusorminus }

BEGIN    { writebools }
  WITH currvals DO
    BEGIN
      goxy( 34, 1 );
      printplusorminus( stat_collect );
      goxy( 44, 1 );
      printplusorminus( data_collect );
      goxy( 57, 1 );
      printplusorminus( regmeasure );
      goxy( 66, 1 );
      printplusorminus( airmeasure );
      goxy( 77, 1 );
      printplusorminus( answerok );
    END;
END;  { writebools }


PROCEDURE writedate;
```

```
{ place the date in the correct place on the screen }

VAR
  s     :       string;

BEGIN
  goxy( 26, 0 );
  date_2_string( date, s );
  prompt( s );
END;  { writedate }


PROCEDURE writetime;
{ update current time }

BEGIN
  goxy( 7, 0 );
  prompt_real( time DIV 100, 2, 0 );
  prompt( ':' );
  prompt_real( time MOD 100, 2, 0 );
  prompt( ':' );
  prompt_real( seconds, 2, 0 );
END;  { writetime }


PROCEDURE writemem;
{ update amount of memory available }

BEGIN
  goxy( 50, 0 );
  prompt_real( memavail, 6, 0 );
END;  { writemem }


PROCEDURE heading;
{ prompts at the top of the main screen }

BEGIN
  clrscrn;
  prompt( 'Time: [        ]   Date: [         ]    ' );
  promptln( 'Memory:  [      ]    Record:   [      ]' );
  prompt( 'Air system [            ]  Stats [ ]  Data [ ]  Regular [ ]' );
  prompt( '  Air [ ]  Phone [ ]' );
END;            { heading }


PROCEDURE footing;
{ prompt at the bottom of the main screen }

BEGIN
  goxy( 0, error_line + 1 );
  clreoln;
  prompt( '<--/--> Select Screen:   R)eturn to Pascal:   Error:  [  ]' );
END;            { footing }


PROCEDURE disperrors;
{ display current error condition }

BEGIN
  goxy( 55, error_line + 1 );
  prompt_real( misc.errors.currerror, 2, 0 );
END;    { disperrors }


PROCEDURE writerecs;
{ update the current workfile record number on the screen }
```

```
BEGIN
  goxy( 72, 0 );
  IF NOT( currvals.data_collect ) THEN prompt( '      ' )
  ELSE prompt_real( recnum, 6, 0 );
END;   { writerecs }


PROCEDURE updatedisplay;

BEGIN
  writetime;
  writedate;
  writemem;
  writeair;
  disperrors;
  writebools;
  writerecs;
  IF currvals.screennum = 2 THEN
    statdisplay( currvals.screennum )
  ELSE IF currvals.screennum = 3 THEN
    statdisplay( currvals.screennum );
END;  { updatedisplay }


PROCEDURE close_modem_files;
BEGIN
  close( outfile );
  close( infile );
  modem_in_use := false;
END;  { close_modem_files }



PROCEDURE open_modem_files;

VAR
  iocode         :        integer;

PROCEDURE test( iocode: integer );

BEGIN
  IF iocode <> 0 THEN
    BEGIN
      modem_in_use := false;
      show_error( iocode );
      exit( open_modem_files );
    END;
END;

BEGIN   { open_modem_files }
  {$I-}
  reset( outfile,'REMOUT:' );
  test( ioresult );
  reset( infile,'REMIN:' );
  test( ioresult );
  {$I+}
  modem_in_use := true;
END;  { open_modem_files }


PROCEDURE checkonline;
{ check to see if carrier is still true; if not then close_modem_files }

BEGIN
  IF NOT carrier THEN
    BEGIN
      unconnect;
      unitclear( 1 );             {flush the type ahead buffer}
      close_modem_files;
    END;
END;    { checkonline }


PROCEDURE goremote;

BEGIN
  goxy( 0, error_line + 1 );
  clreoln;
  evon;
  prompt( 'Output being routed to remote terminal...' );
  evoff;
  open_modem_files;
  heading;
  dispscreen( currvals.screennum );
  footing;
  updatedisplay;
END;            { goremote }


PROCEDURE checkring;
{ look for the phone ringing, answer and open files }

VAR
  i             :       integer;
  phonealert    :       boolean;
  carrcheck     :       boolean;

BEGIN
  IF currvals.answerok THEN
    BEGIN
      i := 0;
      REPEAT
        phonealert := ringing;
        i := succ( i );
      UNTIL phonealert OR ( i > 20 );
      IF phonealert THEN
        BEGIN
          goxy( 0, error_line );
          clreoln;
          prompt( 'Phone ring detected...waiting for carrier ' );
          initcomm( 2 );
          answerthephone;
          IF carrier THEN
            goremote
          ELSE unconnect;
          goxy( 0, error_line );
          clreoln;
        END;      { if phonealert }
    END;        { if answerok }
END;  { checkring }


PROCEDURE checkerror;
{ check the current error condition, see if phone call should be placed }

BEGIN
  WITH misc.errors DO
    IF ( currerror <> noerror ) THEN
      IF ( ( calcsecs( errortime,0 ) > ( errorinterval * 60 ) ) OR
         ( errortime = 0 ) ) AND callok THEN
        call;
END;  { checkerror }


PROCEDURE getgascart( VAR position: integer );
{ reset gascart. if gascart > maxposition then gascart = -1 will measure    }
{ n2 standard gas, gascart = 0 will measure co2 standard gas              }

PROCEDURE findcart( VAR position: integer );
{ scan on-line positions for next position to measure }

BEGIN
  IF position IN [ 1..maxposition ] THEN
    WHILE ( ( position < maxposition ) AND ( gaspos[ position ] < 0 ) ) DO
      position := succ( position );
END;    { findcart }

FUNCTION checkskip: boolean;
{ determine if standard gas measurements should be made }

BEGIN
  currvals.skipcount := succ( currvals.skipcount );
  IF currvals.skipcount > misc.skip THEN
    BEGIN
      currvals.skipcount := 0;
      checkskip := true;
      perform( update_parms( currvals.workvolume, curec ) );
    END
  ELSE
    checkskip := false;
END;    { checkskip }

PROCEDURE handleskip;
{ return either a cart to measure for CO2 or one of the std gas indicators }

BEGIN
  IF checkskip THEN
    position := n2_std
  ELSE
    BEGIN
      position := 1;
      findcart( position );
    END;
END;    { handleskip }

BEGIN  { getgascart }
  position := succ( position );
  IF position IN [ 1..maxposition ] THEN
    findcart( position );
  IF position = maxposition THEN
    IF gaspos[ position ] < 0 THEN
      handleskip
    ELSE
  ELSE IF position > maxposition THEN
    handleskip;
END;    { getgascart }


PROCEDURE domeasure;
{ perform measurement routine, environment control, update screen display }

VAR
  position      :       integer;

BEGIN
  {$R work_stuff}
  FOR position := 1 to max_position DO
    IF ( pos_to_measure[ position ] <> -1 ) THEN
      BEGIN
        measure( position, wrk, recnum, currvals.data_collect );
        IF currvals.stat_collect THEN
          sum_stats( wrk, false );
        adjust( position, wrk );
      END;      { FOR position, IF pos_to_measure... }
  goxy( 0, 2 );
  clreosc;
  dispscreen( currvals.screennum );
  footing;
  updatedisplay;
END;    { domeasure }


PROCEDURE air;
{ perform the correct air operation }

VAR nopurge: boolean;

BEGIN { air }
  CASE currvals.aircondition OF
    gasmeas:    BEGIN
                  nopurge := currvals.gascart = co2_std;
                  { measure, purge lines, reset gascart }
                  airstuff( gasmeas );
                  getgascart( currvals.gascart );
                  nopurge := nopurge OR ( currvals.gascart = co2_std );
                  IF NOT nopurge THEN
                    airstuff( purge )
                  ELSE
                    airstuff( fill );
                END;
    fill   :    airstuff( fill );
    purge  :    airstuff( purge );
  END; { case, if airmeasure }
  writeair;
END;    { air }


PROCEDURE domainedit( VAR pos, chan: integer );
{ edit position and channel number to display on statistics screen         }

VAR
  subscript : integer;
  last_char : char;

PROCEDURE move( subscript: integer );

BEGIN
  IF subscript = 1 THEN
    goxy( 1, error_line )
  ELSE
    goxy( 18, error_line );
END;    { move }

PROCEDURE out( subscript: integer );

BEGIN
  IF subscript = 1 THEN
    prompt_real( pos, 2, 0 )
  ELSE
    prompt_real( chan, 2, 0 );
END;    { out }

PROCEDURE getchange( subscript: integer );

BEGIN
  IF subscript = 1 THEN
    pos := getinteger( pos, maxposition, 1, last_char )
  ELSE
    chan := getinteger( chan, max_field, 1, last_char );
END;    { getchange }

BEGIN   { domainedit }
```

```
  move( 1 );      out( 1 );
  move( 2 );      out( 2 );
  subscript := 1;
  REPEAT
    move( subscript );
    getchange( subscript );
    move( subscript );
    out( subscript );
    IF last_char in [ key_right, key_left ] THEN
      IF subscript = 1 THEN
        subscript := 2
      ELSE
        subscript := 1;
  UNTIL last_char in [ esc, 'Q' ];
END;    { domainedit }


PROCEDURE timeloop;
{ main program operation.  updates time, checks for interrupts and keypress }

VAR
  modemtime,
  modemsecs     :       integer;

BEGIN   { timeloop }
  time := readtime;
  lastsec := seconds;
  modemtime := time;
  modemsecs := seconds;
  REPEAT
    IF modem_in_use THEN
      checkonline               { check to see if the remote end has hung up }
    ELSE
      checkring;                { check to see if the phone is ringing }
    time :=  readtime;
    checkerror;         { check if an unresolved error remains }
    IF seconds <> lastsec THEN
      BEGIN
        lastsec :=  seconds;
        IF modem_in_use THEN
          IF ( calcsecs( modemtime, modemsecs ) > 4 ) THEN
          { update time every 5 seconds over the modem }
            BEGIN
              modemtime := time;
              modemsecs := seconds;
              writetime;
            END
          ELSE
        ELSE writetime;
        IF NOT( currvals.stat_collect ) AND currvals.beep THEN
          IF ( seconds MOD 6 ) = 0 THEN
            BEGIN
              beep; beep;
            END;
        IF measinterrupt THEN
          domeasure;
        IF airinterrupt THEN
          air;
      END;  { if seconds<>lastsec }
  UNTIL keypress;
END; { timeloop }


PROCEDURE cleanwork( vol: integer; s: string );
{ initialize new data file or statistics file, which is the volume number }

VAR
  ch    :       char;
```

```
BEGIN
  goxy( 0, error_line );
  clreoln;
  prompt( concat( 'Remove old ', s, '? [ Y/N ] ' ) );
  ch := getchar( [ 'Y', 'N' ], true, false );
  IF ch = 'Y' THEN
    BEGIN
      IF executed(  get_keys(  vol, keys  )  ) THEN
        BEGIN
          keys.rtotal := 0;
          perform(  update_keys(  vol, keys  )  );

          (* potential code to insert....
          IF executed( openworkfile( vol ) ) THEN
            BEGIN
              seek( workfile, 1 );
              get( workfile );
              close( workfile, crunch );
            END;      { IF executed( openworkfile... }
          *)

        END;
    END;
  goxy( 0, error_line );
  clreoln;
END;    { cleanwork }


PROCEDURE zero_bases;
{ when statistics file is initialized, want all base pointers set to some   }
{ illegal value.  As a result, a  new stat period will be created in the    }
{ stat file the next time statistics is turned on.                          }

VAR
  i     :      integer;

BEGIN
  FOR i := 1 TO maxposition DO
    IF executed( load_parms( currvals.workvolume, i ) ) THEN
      BEGIN
        cart.baseptr := missing;
        perform( update_parms( currvals.workvolume, i ) );
      END;
END;    { zero_bases }


PROCEDURE dooption( ch : char );
{ Process keyboard input, and check condition }

BEGIN
  IF ch = key_up THEN           { undocumented feature                      }
    BEGIN
      currvals.beep := false; { stop no-file beeping }
      perform( update_parms( currvals.workvolume, curec ) );
    END;
  CASE ch OF
      'A':  regtog;             { toggle regular interrupts                 }
      'B':  airtog;             { toggle air interrupts                     }
      'C':  phonetog;           { toggle phone answering                    }
      'D':  datatog;            { toggle data collection                    }
      'E':  statstog;           { toggle statistics collection              }
      'F':  quit( ch );         { go to examine program                     }
      'G':  domeasure;          { take a measurement                        }
      'H':  quit( ch );         { go to test devices program                }

      'I':  quit( ch );         { go to define parameters program           }
      'J':  quit( ch );         { go to edit keyfile program                }
      'K':  quit( ch );         { go to edit labels program                 }
      'L':  quit( ch );         { go to the set'clock program               }
      'M':                      { init new workfile                         }
            cleanwork( currvals.workvolume, 'data file' );
      'N':  BEGIN               { init new stat workfile                    }
              cleanwork( currvals.statvolume, 'statistics file' );
              zero_bases;       { changes all base pointers to missing      }
            END;
      'O':  quit( ch );         { go to the select/merge program            }
      'P':  BEGIN               { edit pos/chan for stat display            }
              ch := 'V';        { update screen afterwards                  }
              goxy( 0, error_line );
              clreoln;
              prompt( '[    ] postion     [    ] channel;' );
              prompt( '  <-/-> move; <esc>' );
              domainedit( misc.pos, misc.chan );
              perform( update_parms( currvals.workvolume, mirec ) );
            END;
      (* 'R':  condition := same; *)
      'S':  quit( ch );         { go to the workfile edit program           }
  END;  { case }

  { take care of screen selection }
  IF ch IN [ key_right, key_left, 'V' ] THEN
    WITH currvals DO
      BEGIN
        IF ch = key_right THEN
          screennum := succ( screennum )
        ELSE IF ch = key_left THEN
          screennum := pred( screennum );
        IF ( screennum > 3 ) OR ( NOT stat_collect ) THEN screennum := 1
        ELSE IF screennum < 1 THEN screennum := 3;
        goxy( 0, 2 );
        clreosc;
        dispscreen( screennum );
        footing;
        updatedisplay;
      END;  { if condition <> same, with currvals ...  }
END;  { dooption }


BEGIN  { main program }
   heading;
   initialize;
   goxy( 0, 2 );
   clreosc;
   dispscreen( currvals.screennum );
   footing;
   updatedisplay;
   REPEAT
      timeloop;
      IF keypress THEN
        BEGIN
          ch := getchar( [ chr( 0 )..chr( 127 ) ], false, false );
          dooption( ch );
        END;    { if keypress }
   UNTIL ( ( NOT modem_in_use ) AND ( ch = 'R' ) );
   clrscrn;
   goxy( 0, 10 );
   prompt( 'Controller program ended... ' );
END.
```

```
{----------------------------------------------}
{ controll.text - an include file for the      }
{ Control program.  Contains the initialize,   }
{ statistics display and the airstuff segment  }
{ procedures.                                  }
{                                              }
{ 11 Oct 85     1600           -mkm-           }
{----------------------------------------------}

{----------------------------------------------}
{ initialize - This routine recovers the old   }
{ program state; sets switches, gas arrays,    }
{ record number, and initializes the display.  }
{----------------------------------------------}


segment PROCEDURE initialize;

VAR
  i: integer;


PROCEDURE readparamfile;
{ load in values from the parameter file, set up on-line cart lists }

VAR
  i: integer;

PROCEDURE period_prompt;

BEGIN
  prompt( '.' );
END;    { period_prompt }

BEGIN { readparamfile }
   goxy( 0, error_line );
   prompt( 'Reading parameter file' );
   IF NOT( executed( load_parms( 4, curec ) ) ) THEN
     BEGIN
       clrscrn;
       goxy( 0, error_line );
       promptln( 'No parameter file available on volume 4.' );
       promptln( 'Fatal error.' );
       goxy( 0, 10 );
       prompt( 'Control program ended......' );
       exit( PROGRAM );
     END;       { if not executed..... }
   FOR i := 1 TO maxposition DO
     BEGIN
       period_prompt;
       IF executed( load_parms( currvals.workvolume, i ) ) THEN
         BEGIN
           IF cart.online THEN
             pos_to_measure[ i ] := cart.cartnum
           ELSE
             pos_to_measure[ i ] := -1;
           IF cart.aironline THEN
             gaspos[ i ] := cart.cartnum
           ELSE
             gaspos[ i ] := -1;
           IF date <> cart.ratetime.lastdate THEN
             WITH cart.ratetime DO
               BEGIN
                 lastdate := date;
                 lasttime := time;
                 lastsecs := seconds;
                 perform( update_parms( currvals.workvolume, i ) );
               END;     { if, with, ... }
         END;   { if executed }
     END;       { for i := 1... }
   period_prompt;
   perform( load_parms( currvals.workvolume, curec ) );
   period_prompt;
   perform( load_parms( currvals.workvolume, mirec ) );
   goxy( 0, error_line );
   clreoln;
   prompt( 'Reading labels' );
   perform( get_labels( currvals.workvolume, labels ) );
END;  { readparamfile }


PROCEDURE recover_state;                { recover old program state }

VAR
  i: integer;

BEGIN
  FOR i := 1 TO maxswitch DO
    IF currvals.switchvals[ i ] THEN switch( i, on, false )
    ELSE switch( i, off, false );
  IF currvals.airmeasure THEN
    IF currvals.aircondition = gasmeas THEN
      WITH currvals DO
        BEGIN
          switch( airpump, on, false );
          gasselect( gascart );
          str( gascart, airstring );
          airstring := concat( 'Fill: pos ', airstring );
          IF length( airstring ) < 12 THEN airstring := concat( airstring, ' ' );
        END;
  IF executed( get_keys( currvals.workvolume, keys ) ) THEN
    recnum := keys.rtotal - 1;
END;  { recover_state }


BEGIN   { initialize }
   writemem;
   goxy( 0, error_line );
   clreoln;
   prompt( 'Initializing' );
   readparamfile;
   recover_state;

   time := readtime;
   airtime := time;
   airseconds := seconds;
   samptime := time;
   sampseconds := seconds;
   goxy( 0, 22 );
   clreosc;
END;    { initialize }

{-----------------------------------------------}
{ statdisplay -                                 }
{ Display the appropriate statistics screen as  }
{ the main menu and fill in using the           }
{ information from the statistics workfile.     }
{ If screen_num = 2, then display one position  }
{ all channels.                                 }
{ If screen_num = 3, then display one channel   }
{ all positions.                                }
{-----------------------------------------------}

segment PROCEDURE statdisplay( VAR menunumber: integer );
```

```
VAR
  field,
  line_number   :       integer;


PROCEDURE linear_display( VAR wrk: work_rec );
{ output linear regression parameters for transperplant }

BEGIN   { leardisplay }
  WITH wrk DO
    BEGIN
      prompt( '       ' );
      prompt_real( data[ nfield ], 5, 0 );    { # of obs. }
      IF data[ nfield ] > 0 THEN
        BEGIN
          prompt( '    ' );
          prompt_real( data[ yintfield ], 8, 2 );       { y-intercept }
          prompt( '       ' );
          prompt_real( data[ slopefield ], 8, 2 );      { slope }
          prompt( '       ' );
          prompt_real( data[ rfield ], 8, 2 );          { corr. coeff. }
          prompt( '       ' );
          prompt_real( data[ errfield ], 8, 2 );        { std. error }
        END;  { if data[ nfield ] ..... }
    END;        { with wrk... }
END;    { linear_display }


PROCEDURE reg_display( VAR wrk: work_rec );
{ write the regular statistics }

BEGIN
  WITH wrk DO
    BEGIN
      prompt( '       ' );
      prompt_real( data[ nfield ], 5, 0 );
      IF data[ nfield ] > 0 THEN
        BEGIN
          prompt( '    ' );
          prompt_real( data[ meanfield ], 8, 2 );
          prompt( '        ' );
          prompt_real( data[ sdfield ], 8, 2 );
          prompt( '       ' );
          prompt_real( data[ cvfield ], 8, 2 );
          prompt( '       ' );
          prompt_real( data[ maximum_field ], 8, 2 );
          prompt( '      ' );
          prompt_real( data[ minimum_field ], 8, 2 );
        END;    { if data[ nfield ] ..... }
    END;        { with wrk do.. }
END;    { reg_display }


PROCEDURE display_rec( VAR wrk: work_rec; VAR line_number: integer );

BEGIN
  WITH wrk DO
    BEGIN
      IF data[ nfield ] > 0 THEN
        BEGIN
          { display the date and the period }
          goxy( 50, 2 );    prompt_real( data[ lastdatefield ], 8, 4 );
          goxy( 69, 2 );
          IF trunc( data[ periodfield ] ) = 3 THEN
            prompt( '        day' )
          ELSE IF trunc( data[ periodfield ] ) = 1 THEN
            prompt( '      night' )
          ELSE prompt( 'transition' );
        END;            { if data[ nfield ] > 0 .... }
    END;    { with wrk.. }
  goxy( 0, line_number );
  IF trunc( wrk.data[ chanfield ] ) = tran_plnt THEN
    linear_display( wrk )                  { display slope, r sesq etc... }
  ELSE
    reg_display( wrk );                 { display regular stats }
  promptln( '' );
  line_number := line_number + 1;
END;    { display_rec }


PROCEDURE do_position;
{ display one position, all channels                                        }

VAR
  position,
  base,
  i     :       integer;

BEGIN
  position := misc.pos;
  perform( load_parms( currvals.workvolume, position ) );
  base := cart.base_ptr;
  goxy( 10, 2 );            prompt_real( position, 2, 0 );
  goxy( 22, 2 );            prompt_real( cart.cartnum, 2, 0 );
  line_number := 5;
  FOR i := 1 TO max_field DO
    IF misc.collect_stats[ i ] THEN
      BEGIN
        IF executed( get_rec( base, wrk ) ) THEN
          display_rec( wrk, line_number );
        base := base + 1;
      END;      { for, if }
END;    { do_position }


PROCEDURE do_channel;

VAR
  channel,
  base,
  i     :       integer;

FUNCTION find_rec( base, chan: integer ): integer;
{ calculater the appropriate record from the channel-position }

VAR
  i     :       integer;

BEGIN
  i := 1;
  WHILE i < chan DO
    BEGIN
      IF misc.collect_stats[ i ] THEN
        base := base + 1;
      i := i + 1;
    END;
  find_rec := base;
END;    { find_rec }


BEGIN   { do_channel }
  channel := misc.chan;
  goxy( 9, 2 );             prompt_real( channel, 2, 0 );
  goxy( 21, 2 );            prompt( labels[ channel ] );
  line_number := 5;
  IF misc.collect_stats[ channel ] THEN
    FOR i := 1 TO max_field DO
      BEGIN
        perform( load_parms( currvals.workvolume, i ) );
        IF cart.online THEN
          BEGIN
            base := find_rec( cart.base_ptr, channel );
            IF executed( get_rec( base, wrk ) ) THEN
              display_rec( wrk, line_number );
          END;
      END;      { if, for.. }
END;    { do_channel }


BEGIN   { statdisplay }
  {$R workstuff}
  IF executed( open_workfile( currvals.statvolume ) ) THEN
    IF executed( get_keys( currvals.statvolume, keys ) ) THEN
      BEGIN
        IF menu_number = 2 THEN
          do_position
        ELSE IF menu_number = 3 THEN
          do_channel;
      END;      { if executeds... }
  close_workfile;
END;    { statdisplay }

{-----------------------------------------------}
{ Air_stuff - The set of air system procedures. }
{-----------------------------------------------}

segment PROCEDURE airstuff( whattodo: airtype );


PROCEDURE co2control( position: integer );

PROCEDURE co2squirt( delco2, volume, flowrate: real; position: integer );

{---------------------------------------------------------------}
{ Automatically brings the level in the cart up to optimum.     }
{ Volume is the chamber hood volume in liters.  Delco2 is the   }
{ difference between desired co2 level and actual in ppm.       }
{ Flowrate is the rate CO2 is delivered from the bottle in      }
{ cc/sec.                                                       }
{                                                               }
{ Logic description:  The idea is that knowing how low the      }
{ CO2 level is in the chamber, and the rate at which the CO2    }
{ will be delivered, the time that is necessary for the valve   }
{ to be opened can be determined.                               }
{                                                               }
{ 1.  Calculate the amount of CO2 to be added, in milliliters   }
{ ( cc ), by:  amount of CO2 in 1 million parts of air times    }
{ the chamber volume in ml.                                     }
{                                                               }
{ 2.  The length of time to open the valve is calculated by     }
{ dividing the flow rate at which CO2 is delivered into the     }
{ mls of CO2 which must be added.  The time is in seconds.      }
{---------------------------------------------------------------}

VAR
  addco2        :       real;

BEGIN   { co2squirt }
  goxy( 0, error_line );
  clreoln;
  prompt( 'Adding co2 to cart ' );
  prompt_real( position, 2, 0 );
  addco2 := ( delco2 * volume / 1000 );         {1} { mls calculated }
  gasselect( position );
  switch( squirt, on, false );
  stopwatch( addco2 / flowrate );              {2} { time calculated }
  switch( squirt, off, false ),
  stopwatch( lagtime );
  gasselect( off_cart );
  goxy( 0, error_line );
  clreoln;
END;  { co2squirt }


PROCEDURE turnoffco2( position: integer );

BEGIN
  cartselect( position );
  switch( co2off, on, false );                  { turns co2 flow off }
  stopwatch( 0.25 );
  switch( co2off, off, false );
  cartselect( off_cart );
END;  { turnoffco2 }


PROCEDURE turnonco2( position: integer );

BEGIN
  cartselect( position );
  switch( co2on, on, false );
  stopwatch( 0.25 );
  switch( co2on, off, false );
  cartselect( off_cart );
END;  { turnonco2 }


PROCEDURE setrate( VAR rate: real; position: integer );

{-----------------------------------------------------------------}
{ Set the rate on the CO2 mass flow controller.  Return the      }
{ rate that was set for checking purposes.  Rate is the rate     }
{ to set, in ppm/min ( actually, cc/min )                        }
{-----------------------------------------------------------------}

VAR
  volts : real;
  dacnum: integer;

BEGIN
  volts := 0.25 * rate;
  (* volts := 0.0429 + 0.4519 * rate;  { this is the 10 cc/min flow eqn } *)
  IF volts > 5.0 THEN
    volts := 5.0
  ELSE IF volts < 0.1 THEN
    volts := 0.1;
  dacnum := round( -10.4286 + 104.2857 * volts );

  { recreate rate so you can see the difference between actual and desired }
  { rates }
  volts :=  0.1 + ( 0.0096 * dacnum );
  rate := 4.0 * volts;
  (* rate := -0.095 + ( 2.213 * volts ); { the old 10 cc/min controller } *)

  cartselect( position );
  dac( dacnum );
  cartselect( off_cart );
END;  { setrate }


PROCEDURE fixco2flow( delco2, diffco2: real; position: integer );
```

```
{-------------------------------------------------------------}
{ Calculate what the new rate on the CO2 mass flow controller }
{ should be.                                                  }
{                                                             }
{ Delco2 is the difference between lastco2 and this co2       }
{ measurement.  Diffco2 is the difference between optimum co2 }
{ and this co2 measurement.  The rates in this procedure      }
{ should be in ppm/second, flowrate in cc/sec.  Note that the }
{ rates can also be translated as cc/sec.                     }
{                                                             }
{ Logic description:  The objective in this routine is to     }
{ adjust the continual CO2 flow into the chamber in order to  }
{ bring it to the optimum concentration the next time the     }
{ chamber is measured.  The estimated rate at which the       }
{ plants are using CO2, the photosynthetic rate, is also      }
{ calculated.                                                 }
{                                                             }
{ 1.  Calculate the rate of change which is needed to bring   }
{ the chamber to equilibrium by the time of the next          }
{ measurement.                                                }
{                                                             }
{ 2.  Calculate the rate of change which occurred between the }
{ last measurement and this measurement ( the observed rate   }
{ of change ).                                                }
{                                                             }
{ 3.  The photosynthetic rate, the rate at which the plants   }
{ are using the CO2 in the chamber, is the observed rate of   }
{ change minus the current CO2 injection rate.                }
{                                                             }
{ 4.  Then, in order to bring the chamber to equilibrium by   }
{ the next next measurement, add the rate of change which is  }
{ needed and the photosynthetic rate to get the new CO2       }
{ injection rate.                                             }
{-------------------------------------------------------------}

VAR
  deltime,
  estrate,        { rate of change needed to bring cart to equibilibrium }
  lastrate,       { observed rate of change since last measurement }
  photorate,      { estimated photosynthetic rate }
  newrate         :       real;   { total CO2 rate to bring cart to equilib. }

BEGIN
  turnonco2( position );
  time := readtime;
  WITH cart.ratetime DO
    deltime := calcsecs( lasttime, lastsecs );
  deltime := deltime / 60;        { all times in minutes }

  { liters * ppm / min / ( *1000 cc per liter / 1000000 cc = 1000 ) }
  estrate := ( cart.hoodvol * diffco2 / deltime ) / 1000;              (1)

  lastrate := ( cart.hoodvol * delco2 / deltime ) / 1000;              (2)

  photorate := cart.currrate - lastrate;                               (3)
  currvals.gasarray[ position ].photodata := photorate;

  newrate := estrate + photorate;                                      (4)

  setrate( newrate, position );
  cart.currrate := newrate;
  cart.lastco2 := currvals.gasarray[ position ].co2data;
  IF diffco2 < -cart.co2info.span THEN
    turnoffco2( position )
  ELSE IF diffco2 > cart.co2info.span THEN
    BEGIN
      co2squirt( diffco2, cart.hoodvol, misc.flowrate, position );
      cart.lastco2 := cart.co2info.optimum;
    END;

  cart.ratetime.lasttime := time;
  cart.ratetime.lastsecs := seconds;
  cart.ratetime.lastdate := date;
END;  { fixco2flow }


BEGIN  { co2control }
  IF ( cart.lightinfo > cart.last_lite ) THEN  { lights are off    }
    BEGIN
      IF cart.newair THEN
        WITH cart.co2info, currvals.gasarray[ position ] DO
          fixco2flow( co2data - cart.lastco2, optimum - co2data, position );
      turnoffco2( position );
    END
  ELSE                                               { lights are on     }
    WITH cart.co2info, currvals.gasarray[ position ] DO
      IF cart.newair THEN
        fixco2flow( co2data - cart.lastco2, optimum - co2data, position )
      ELSE
        BEGIN
          IF ( optimum + span < co2data ) THEN       { conc. is too high }
            turnoffco2( position )
          ELSE
            IF ( optimum - span > co2data ) THEN     { conc. is too low  }
              BEGIN
                co2squirt( optimum - co2data,
                           cart.hoodvol,
                           misc.flowrate, position );
                turnonco2( position );
              END       { if optimum - span > co2data }
        END;  { with cart, co2info }
END;  { co2control }


PROCEDURE purgelines;
{ Set system to purge the air lines with outside air }

BEGIN
  currvals.airstring := 'Purging      ';
  switch( airswitch, on, false );
  switch( airpump, on, false );
  switch( vent, on, true );
  currvals.aircondition := fill;
END;  { purgelines }


PROCEDURE gasmeasure( position: integer );
{ After lines have been filled with chamber air, measure CO2 content }

VAR
  i,
  iocode         :       integer;
  vr             :       voltrange;

{ gasmeasurements are stored in an array[ -1..maxposition ] of  }
{ a record which holds a boolean flag and two real data values. }
{ -1 = n2 standard, 0 = co2 standard.                           }

BEGIN  { gasmeasure }

  { first, get co2 measurement }
  vr := misc.channels[ co2_chan ];
  currvals.gasarray[ position ].co2data := integratea2d( co2_chan, vr );

  IF position >= 1 THEN
    WITH currvals DO
```

```
      BEGIN
        perform( load_parms( currvals.workvolume, position ) );
        volts_2_scientific( co2_chan,
                            cart.conversions,
                            gasarray[ position ].co2data );
        { then, the humidity }
        vr := misc.channels[ h2ochan ];
        gasarray[ position ].h2odata := integratea2d( h2ochan, vr );
        gasarray[ position ].time := time_2_r( time, seconds );
        volts_2_scientific( h2o_chan,
                            cart.conversions,
                            gasarray[ position ].h2o_data );
        co2control( position );      { also saves photodata in gasarray }
        gasselect( off_cart );
        perform( update_parms( currvals.workvolume, position ) );
      END;

  IF position < 1 THEN { taking calibration measurements }
    BEGIN
      switch( n2gas, off, false );
      switch( co2stdgas, off, false );
      switch( airswitch, off, false );
      switch( vent, off, false );
    END;  { if position < 1 }

  switch( airpump, off, true );
  currvals.aircondition := purge;

END;  { gasmeasure }


PROCEDURE filllines( position: integer );
{ fills lines with gas from correct cart or standard gas bottle }

BEGIN
  IF position = co2_std THEN  { fill with co2 standard }
    BEGIN
      currvals.airstring := 'Fill: CO2   ';
      switch( vent, on, false );
      switch( airswitch, off, false );
      switch( co2stdgas, on, false );
      switch( airpump, off, true );
    END

  ELSE IF position = n2_std THEN { fill with n2 standard }
    BEGIN
      currvals.airstring := 'Fill: N2    ';
      switch( vent, on, false );
      switch( airswitch, off, false );
      switch( n2gas, on, false );
      switch( airpump, off, true );
    END

  ELSE { fill with co2 from cart. }
    WITH currvals DO
      BEGIN
        str( position, airstring );
        airstring := concat( 'Fill: pos ', airstring );
        WHILE length( airstring ) < 12 DO
          airstring := concat( airstring, ' ' );
        switch( vent, off, false );
        switch( airswitch, off, false );
        switch( airpump, on, true );
        gasselect( position );
      END;
  currvals.aircondition := gasmeas;
END;  { filllines }
```

```
BEGIN { airstuff }
  CASE whattodo OF
    purge: purgelines;
    fill:  filllines( currvals.gascart );
    gasmeas: gasmeasure( currvals.gascart );
  END;
  airtime := readtime;
  airseconds := seconds;
  perform( update_parms( currvals.workvolume, curec ) );
END;  { airstuff }
```

```
{-------------------------------------------------------}
{  control2 - an include file for the Control program.  }
{  This file contains the measure and the statistics    }
{  summation segment procedures.                        }
{                                                       }
{  23 Oct 85            1600            -mkm-           }
{-------------------------------------------------------}


{-------------------------------------------------------}
{  Segment Procedure measure.                           }
{  This procedure takes a measurement from the cart     }
{  specified, stores the new record on the workfile,    }
{  updates the cart parameters, and passes back the new }
{  record so that the cart can be environmentally       }
{  controlled.                                          }
{-------------------------------------------------------}

segment PROCEDURE measure( position: integer;         { position to measure  }
                           VAR wrk : workrec;         { the workfile record  }
                           VAR rec : integer;         { record number        }
                           store_it: boolean );       { save record ?        }

VAR
  field_num     :       integer;
  ch            :       char;


FUNCTION rate_calc( pos : integer;              { position of interest      }
                    VAR wrk: workrec )          { to get tran soln info     }
                    : real;
{ calculate the transpired solution rate for this position }
{ transp. soln / plant / minute }

VAR
  i                             :       integer;
  sx, sxsqr, sxy, sy, sysqr, n  :       real;

BEGIN
  sx := 0; sxsqr := 0; sxy := 0; sy := 0; sysqr := 0; n := 0;
  WITH cart DO
    BEGIN

      { adjust the transpired rate array to accept next measurement }
      moveleft( trans_rate[ 2 ], trans_rate[ 1 ], 70 );

      WITH trans_rate[ misc.numpoints ] DO
        BEGIN
          { add the latest measurement to the array for next calculation }
          value := wrk.data[ tran_plnt ];
          abstime.lasttime := time;
          abstime.lastsecs := seconds;
          abstime.lastdate := date;
        END;     { with trans_rate ...... }
    END;        { with currvals .... }

    { check to see if the array is full }
    WITH cart.trans_rate[ 1 ] DO
      IF ( value = 0 ) AND ( abstime.lasttime = 0 ) AND
                                                 ( abstime.lastsecs = 0 ) THEN
        BEGIN
          rate_calc := missing;
          exit( rate_calc );
        END;
    { now, sum the values and calculate the slope }

    FOR i := 1 TO misc.numpoints DO
```

```
      WITH cart.trans_rate[ i ] DO
        IF NOT ( ( abstime.lasttime = 0 ) AND ( abstime.lastsecs = 0 ) AND
                                                           ( value = 0 ) ) THEN
          BEGIN
            etime := 60 * ( elapsed_time( lastdate, lasttime, lastsecs,
                            cart.transrate[ 1 ].lastdate,
                            cart.transrate[ 1 ].lasttime,
                            cart.transrate[ 1 ].lastsecs ) );
            n := n + 1;
            sy := sy + value;
            sysqr := sysqr + ( value * value );
            sx := sx + etime;
            sxsqr := sxsqr + ( etime * etime );
            sxy := sxy + ( value * etime );
          END;    { for i := ...... }

    rate_calc := slope ( sx, sxsqr, sxy, sy, n );
END;    { rate_calc }


PROCEDURE get_field( field,                    { field number in workfile  }
                     position: integer;        { position for reading      }
                     VAR wrk: workrec;         { has impt info sometimes   }
                     VAR data: real );         { the answer                }
{ Inputs the measured values into the workfile                             }


PROCEDURE get_reading( field: integer; vr: voltrange; VAR data: real );
{ gets reading and converts to real numbers }

BEGIN
  IF field IN [ 1..maxchan ] THEN
    BEGIN
      data := integratea2d( field, vr );
      volts_2_scientific( field, cart.conversions, data );
    END;
END;    { get_reading }


PROCEDURE get_lite( position, lite_pos: integer; VAR data: real );
{ get light reading from the designated light position                     }

BEGIN
  cart_select( lite_pos );
  stopwatch( 0.25 );
  get_reading( lite_chan, misc.channels[ lite_chan ], data );
  cart_select( position );
  stopwatch( 0.25 );
END;    { get_lite }


FUNCTION do_elap_time( e_date: ucsd_date; e_time, e_secs: integer ): real;

BEGIN
  do_elap_time := elapsedtime( date, time, seconds, edate, etime, esecs );
END;    { do_elap_time }


BEGIN   { get_field }
  CASE field OF
    1, 2, 3, 4, 5, pH_chan, soln_chan, 10, co2_flow, 15, 16:
      get_reading( field, misc.channels[ field ], data );
    lite_chan :                                { get light reading         }
      BEGIN
        IF position = cart.lite_pos THEN
          get_reading( field, misc.channels[ field ], data )
        ELSE
          get_lite( position, cart.lite_pos, data );
    cart.last_lite := data;
  END;      { lite_chan }
tran_chan :                                 { transpired solution data, }
                                            { sum total soln transpired }
  BEGIN
    get_reading( tran_chan, misc.channels[ tran_chan ], data );
    cart.tottranspired := cart.tottranspired + data;
  END;
elec_chan  :                                { electrical conductivity   }
                                            { delay avoids complications}
                                            { with the pH measurements  }
  BEGIN
    switch( elecswitch, on, false );
    stopwatch( 2.5 );
    get_reading( elec_chan, misc.channels[ elec_chan ], data );
    switch( elecswitch, off, false );
  END;  { elec_chan }
co2_chan   :                                { read co2 from storage     }
                                            { array - reset to missing  }
  WITH currvals.gasarray[ position ] DO
    BEGIN
      data := co2data;
      co2data := missing;
    END;
h2o_chan   :                                { read water vapor from     }
                                            { storage array - reset     }
  WITH currvals.gasarray[ position ] DO
    BEGIN
      data := h2odata;
      h2odata := missing;
    END;
tot_soln  :                                 { total solution added this }
                                            { period                    }
  data := cart.addedtoday;
tran_plnt:                                  { transpired soln per plant }
  WITH cart DO
    IF ( numplants <> 0 ) AND ( tottranspired <> missing ) THEN
      data := tottranspired / numplants;
phot_plnt:                                  { photosynthetic rate per   }
                                            { plant.                    }
  WITH cart, currvals.gasarray[ position ] DO
    BEGIN
      IF ( numplants <> 0 ) AND ( photodata <> missing ) THEN
        data := photodata / numplants;
      photodata := missing;
    END;
pos_cart:                                   { the position.cart info    }
  data := position + ( pos_to_measure[ position ] / 100 );
date_fld :                                  { date - MM.DDYY            }
  data := date_2_r( date );
time_fld :                                  { time - HH.MMSS            }
  data := time_2_r( time, seconds );
co2_time:                                   { time of co2 measurement   }
  WITH currvals.gasarray[ position ] DO
    BEGIN
      data := time;
      time := missing;
    END;
n2_fld   :                                  { n2 std measurement        }
  BEGIN
    data := currvals.gasarray[ n2_std ].co2data;
    currvals.gasarray[ n2_std ].co2data := missing;
  END;
co2_fld  :                                  { co2 std measurement       }
  BEGIN
    data := currvals.gasarray[ co2_std ].co2data;
    currvals.gasarray[ co2_std ].co2data := missing;
  END;
```

```
    std_time:                                { time of std measurement    }
      WITH currvals DO
        IF gasarray[ n2_std ].time <> missing THEN
          BEGIN
            data := currvals.gasarray[ n2_std ].time;
            currvals.gasarray[ n2_std ].time := missing;
          END
        ELSE IF gasarray[ co2_std ].time <> missing THEN
          BEGIN
            data := gasarray[ co2_std ].time;
            gasarray[ co2_std ].time := missing;
          END;
    elap_time:                               { elapse time, since dosage }
      IF cart.dosed THEN
        WITH cart.dosetime DO
          data := elapsedtime( date, time, seconds,
                                             lastdate, lasttime, lastsecs )
      ELSE
        data := missing;
    plnt_fld  :                              { number of plants          }
      data := cart.numplants;
    tran_rate:                               { transpiration rate        }
      data := rate_calc( position, wrk );
  END;  { case }
END;    { get_field }


PROCEDURE display_field( field: integer; data: real );
{ display the newly measured field }

VAR
  x     :       integer;

BEGIN
  IF field IN [ 1..12 ] THEN
    x := 14
  ELSE IF field IN [ 13..24 ] THEN
    x := 39
  ELSE
    x := 64;
  WHILE field > 12 DO
    field := field - 12;
  goxy( x, field + 5 );
  IF data <> missing THEN
    prompt_real( data, 8, 4 )
  ELSE
    prompt( '         ' );                   { nine spaces               }
END;    { display_field }


PROCEDURE disp_labels;
{ display the current labels on the screen }

VAR
  i     :       integer;

BEGIN
  FOR i := 1 TO maxfield DO
    BEGIN
      IF i IN [ 1..12 ] THEN
        goxy( 0, i + 5 )
      ELSE IF i IN [ 13..24 ] THEN
        goxy( 25, i - 7 )
      ELSE
        goxy( 50, i - 19 );
      prompt_real( i, 2, 0 );
      prompt( '. ' );
      prompt( labels[ i ] );
    END;        { for i... }
END;    { disp_labels }


BEGIN  { measure }
  cart_select( off_cart );                      { disable all positions     }
  perform( load_parms( currvals.workvolume, position ) );
                                                { load parameters for cart  }
  cart_select( position );                      { enable this position      }

  goxy( 0, 2 );                                 { screen display stuff      }
  clreosc;                                      {          |                }
  prompt( 'Measurement series:    Position:       Cart:' );
  disp_labels;                                  {          |                }
  promptat( 0, error_line + 1 , 'W )ait;  <esc> return to Main menu' );
  writemem;                                     {          |                }
  time := readtime;                             {          |                }
  writetime;                                    {          |                }
  writedate;                                    {          |                }
  goxy( 33, 2 );                                {          |                }
  prompt_real( position, 2, 0 );                {          |                }
  goxy( 45, 2 );                                {          |                }
  prompt_real( cart.cartnum, 2, 0 );            {          |                }

  switch( transoff, on, false );                { turn transpired dump off  }
  stopwatch( 0.25 );                            { by toggling switch        }
  switch( transoff, off, false );

  zero_wrk( wrk );                              { zero workfile record      }

  stopwatch( 1.0 );                             { wait one second           }

  FOR field_num := 1 TO max_field DO            { measure and display       }
    BEGIN
      get_field( field_num, position, wrk, wrk.data[ field_num ] );
      IF ( labels[ field_num ] <> '         ' ) THEN
        display_field( field_num, wrk.data[ field_num ] );
    END;


  IF store_it THEN                              { save record on workfile   }
    BEGIN
      IF executed( get_keys( currvals.workvolume, keys ) ) THEN
        IF keys.rtotal < max_per_disk THEN
          IF executed( openworkfile( currvals.workvolume ) ) THEN
            IF executed( append_rec( keys.rtotal, wrk ) ) THEN
              IF executed( update_keys( currvals.workvolume, keys ) ) THEN
                rec := keys.rtotal - 1;
      close_workfile;
    END;        { if store_it }

  perform( update_parms( currvals.workvolume, position ) );
  perform( update_parms( currvals.workvolume, curec ) );

  IF keypress THEN                              { check for wait            }
    BEGIN
      ch := getchar( [ chr( 0 )..chr( 127 ) ], false, false );
      IF ch = 'W' THEN
        BEGIN
          goxy( 0, error_line + 2 );
          clreoln;
          evon;
          REPEAT
            goxy( 0, error_line + 2 );
            prompt( 'Press any key to continue' );
          UNTIL keypress;
          evoff;
          ch := getchar( [ chr( 0 )..chr( 127 ) ], false, false );
          ch := space;
          goxy( 0, error_line + 2 );
          clreoln;
        END;
    END;  { if keypress }

  cart_select( off_cart );                      { disable the position      }
  stopwatch( 1.0 );                             { wait for one second       }
END;  { measure }

{-------------------------------------------------------}
{  Segment Procedure sum_stats.                         }
{  This procedure adds the newly measured record to     }
{  the statistics workfile.  All statistics file        }
{  handling is performed by this procedure;  file       }
{  expansion, period checking, as well as statistical   }
{  summing.                                             }
{-------------------------------------------------------}


segment PROCEDURE sum_stats( VAR wrk: workrec; force_new: boolean );


VAR
  w     :       workrec;
  i,
  position,
  rekord:       integer;


{-------------------------------}
{  Statistical functions        }
{-------------------------------}

FUNCTION mean( sum, n: real ): real;
{ calculate the mean                                                        }

BEGIN
  IF n > 0 THEN
    mean := sum / n
  ELSE
    mean := missing;
END;    { mean }


FUNCTION sd( sum, sumsqr, n: real ): real;
{ calculate the standard deviation                                          }

VAR
  ssqrd: real;

BEGIN
  sd := missing;
  IF n > 1 THEN
    BEGIN
      ssqrd := ( sumsqr - ( sqr( sum )/n ) ) / ( n-1 );
      IF ssqrd > 0 THEN
        sd := sqrt( ssqrd );
    END;
END;    { sd }

FUNCTION cv( meen, sdev: real ): real;
{ calculate the coefficient of variation                                    }
{ pass the mean and the standard deviation.                                 }

BEGIN
  cv := missing;
  IF ( meen <> missing ) AND ( sdev <> missing ) THEN
```

```
    IF meen <> 0 THEN
      cv := sdev * 100.0 / meen;
END;    { cv }


FUNCTION yintercept( cslope, sy, sx, n: real ): real;
{ calculate the y intercept of a line                                      }
{ pass calculated slope, sum y's, sum x's, n.                              }

BEGIN
  IF ( n > 0 ) AND ( cslope <> missing ) THEN
    yintercept := ( sy - ( cslope * sx ) ) / n
  ELSE
    yintercept := missing;
END;    { yintercept }


FUNCTION r( sy, sysqr, sx, sxy, cslope, n: real ): real;
{ calculate r - how well the line fits the data.                           }
{ pass: sum y's, sum y's sqrd, sum x's, sum x * y, calculated slope, n.    }

VAR
  value   :       real;

BEGIN
  r := missing;
  IF n > 0 THEN
    BEGIN
      value := sysqr - sqr( sy );
      IF ( value <> 0 ) AND ( cslope <> missing ) THEN
        BEGIN
          value := ( cslope * ( sxy - sx * sy / n ) ) / value / n ;
          IF value > 0 THEN
            r := sqrt( value );
        END;
    END;
END;    { r }


FUNCTION sse( sy, sysqr, sx, sxy, cslope, n: real ): real;
{ calculate the sum of squared error - line error calculation              }

VAR
  value :       real;

BEGIN
  sse := missing;
  IF ( n > 2 ) AND ( cslope <> missing ) THEN
    BEGIN
      value := ( sysqr - sqr( sy ) / n ) - ( cslope * ( sxy - sx * sy / n ) );
      IF value > 0 THEN
        sse := sqrt( value / ( n - 2 ) );
    END;
END;    { sse }


FUNCTION find_rec( base, chan: integer ): integer;
{ Calculates the appropriate record from the channel-position }

VAR
  i     :       integer;

BEGIN
  i := 1;
  WHILE i < chan DO
    BEGIN
      IF misc.collect_stats[ i ] THEN
        base := base + 1;
      i := i + 1;
    END;
  find_rec := base;
END;    { find_rec }


{------------------------------}
{ do_stats, Main procedure in  }
{ this segment.  Performs      }
{ actual summing.              }
{------------------------------}

PROCEDURE do_stats( VAR w: workrec; value: real; channum, position: integer );

VAR
  timeone,
  secsone       : integer;
  dateone       : ucsd_date;
  etime         : real;

BEGIN   { do_stats }
  WITH w DO
    BEGIN
      data[ nfield ] := data[ nfield ] + 1;
      data[ sumy ] := data[ sumy ] + value;
      data[ sumysqr ] := data[ sumysqr ] + sqr( value );
      data[ maximum_field ] := max( data[ maximum_field ], value );
      data[ minimum_field ] := min( data[ minimum_field ], value );
      data[ pos_cart ] := position + ( pos_to_measure[ position ] / 100 );
      data[ chanfield ] := channum;
      data[ lasttimefield ] := time_2_r( time, seconds );
      data[ lastdatefield ] := date_2_r( date );
      data[ periodfield ] := ord( currvals.currperiod );
      r_2_date( data[ firstdatefield ], dateone );
      r_2_time( data[ firsttimefield ], timeone, secsone );
      IF ( date_one.day <> 0 ) AND
         ( date_one.mo <> 0 ) AND
         ( date_one.day <> 0 ) AND
         ( timeone <> 0 ) AND ( secsone <> 0 ) THEN
         etime :=
           elapsedtime( date, time, seconds, dateone, timeone, secsone )
       ELSE
         etime := missing;
       IF etime <> missing THEN
         BEGIN
           data[ sumx ] := data[ sumx ] + etime;
           data[ sumxsqr ] := data[ sumxsqr ] + sqr( etime );
           data[ sumxy ] := data[ sumxy ] + ( etime * value );
         END;

     data[ meanfield ] := mean( data[ sumy ], data[ nfield ] );

     data[ sdfield ] := sd( data[ sumy ], data[ sumysqr ], data[ nfield ] );
     data[ cvfield ] := cv( data[ meanfield ], data[ sdfield ] );

     data[ slopefield ] := slope( data[ sumx ], data[ sumxsqr ], data[ sumxy ],
                                             data[ sumy ], data[ nfield ] );

     data[ yintfield ] := yintercept( data[ slopefield ],
                                 data[ sumy ], data[ sumx ], data[ nfield ] );

     data[ rfield ] := r( data[ sumy ], data[ sumysqr ],
            data[ sumx ], data[ sumxy ], data[ slopefield ], data[ nfield ] );

     data[ errfield ] := sse( data[ sumy ], data[ sumysqr ],
            data[ sumx ], data[ sumxy ], data[ slopefield ], data[ nfield ] );

    END;    { with w do...  }
```

```
END;            { do_stats }


{-------------------------------}
{ check_period -                }
{ This procedure compares the   }
{ Statistic's file period with  }
{ the program's period.  If     }
{ they are not the same, the    }
{ statistics file is updated,   }
{ and a new period of statistics}
{ for this position is begun.   }
{-------------------------------}

PROCEDURE check_period( force_new: boolean );

VAR
  w     :       workrec;
  i     :       integer;

PROCEDURE zero_stat_rec( VAR w: workrec; channel: integer );
{ set the appropriate fields in this record to zero ( instead of missing )  }

BEGIN
  zerowrk( w );
  WITH w DO
    BEGIN
      data[ chanfield ] := channel;
      data[ firsttimefield ] := time_2_r( time, seconds );
      data[ firstdatefield ] := date_2_r( date );
      data[ nfield ] := 0.0;
      data[ sumy ] := 0.0;
      data[ sumysqr ] := 0.0;
      data[ maximum_field ] := minreal;
      data[ minimum_field ] := maxreal;
      data[ periodfield ] := ord( currvals.currperiod );
      data[ sumx ] := 0.0;
      data[ sumxsqr ] := 0.0;
      data[ sumxy ] := 0.0;
      data[ lastdatefield ] := data[ firstdatefield ];
      data[ lasttimefield ] := data[ firsttimefield ];
    END;        { with w do }
END;    { zero_stat_rec }

BEGIN   { check_period }
  IF executed( open_workfile( currvals.statvolume ) ) THEN
    IF executed( get_keys( currvals.stat_volume, keys ) ) THEN
      IF executed( get_rec( cart.baseptr, w ) ) THEN
        IF ( ord( currvals.currperiod ) <> trunc( w.data[ periodfield ] ) ) OR
             force_new THEN
          BEGIN                             { must expand file           }
            goxy( 0, error_line + 2 );
            clreoln;
            prompt( 'Adding new period to statistics file for position ' );
            prompt_real( trunc( wrk.data[ pos_cart ] ), 2, 0 );
            cart.baseptr := keys.rtotal;
            perform( update_parms( currvals.workvolume, position ) );
            FOR i := 1 TO max_field DO
              IF misc.collect_stats[ i ] THEN
                BEGIN
                  zero_stat_rec( w, i );
                  IF NOT( executed( append_rec( keys.rtotal, w ) ) ) THEN
                    BEGIN
                      currvals.stat_collect := false;
                      perform( update_parms( currvals.workvolume, curec ) );
                      exit( sum_stats );
                    END;        { if not executed... }
                END;    { for, if misc.collect_stats... }
            perform( update_keys( currvals.statvolume, keys ) );
          END;  { four if's.... }
  close_workfile;
END;    { check_period }


BEGIN   { sum_stats }
  {$R transcend}
  {$R work_stuff}
  position := trunc( wrk.data[ pos_cart ] );
  IF position IN [ 1..maxposition ] THEN
    BEGIN
      IF executed( load_parms( currvals.workvolume, position ) ) THEN
        BEGIN
          IF cart.base_ptr <> missing THEN
            BEGIN
              check_period( force_new );
              goxy( 0, error_line + 2 );
              clreoln;
              prompt( 'Summing statistics for position ' );
              prompt_real( trunc( wrk.data[ pos_cart ] ), 2, 0 );
              IF executed( open_workfile( currvals.stat_volume ) ) THEN
                IF executed( get_keys( currvals.stat_volume, keys ) ) THEN
                  BEGIN
                    FOR i := 1 TO max_field DO
                      IF misc.collect_stats[ i ] THEN
                        IF wrk.data[ i ] <> missing THEN
                          BEGIN
                            rekord := find_rec( cart.base_ptr, i );
                            IF executed( get_rec( rekord, w ) ) THEN
                              BEGIN
                                do_stats( w, wrk.data[ i ], i, position );
                                perform( update_rec( rekord, w ) );
                              END;        { if executed... }
                          END;      { if wrk.data <> missing... }
                  END;    { if executed( open_workfile, get_keys ).... }
              goxy( 0, error_line + 2 );
              clreoln;
              close_workfile;
            END;        { if cart.base_ptr <> missing ... }
        END;    { if executed( load_parms )... }
    END;        { if position in... }
END;    { sum_stats }
```

```
{-------------------------------------------------------}
{  Control3 - an include file for the Control program.  }
{  This file contains the environmental adjust, quit,   }
{  and the telephone alert segment procedures.          }
{                                                       }
{  11 Oct 85           1030            -mkm-            }
{-------------------------------------------------------}


{-------------------------------------------------------}
{  Segment Procedure adjust.                            }
{  This procedure adjusts the environment of the        }
{  chamber specified.  It controls pH, transpired       }
{  solution, and hydroponic solution.  CO2 level is     }
{  controlled in the airstuff segment.                  }
{-------------------------------------------------------}

segment PROCEDURE adjust( position: integer;    { the position to adjust    }
                          VAR wrk : workrec );  { based on this information }


PROCEDURE soln_check( position: integer; VAR wrk: workrec );
{ check solution level, add if necessary.                               }

VAR
  add   : boolean;


PROCEDURE pump( position, numpumps: integer );
{ deliver this many pumps of hydroponic solution to the position        }

VAR
  x : integer;
  ch: char;

BEGIN
  ch := space;
  x := 0;
  goxy( 0, error_line + 2 );
  clreoln;
  prompt( 'Adding solution to position #' );
  prompt_real( position, 3, 0 );
  switch( solenoid, on, false );
  switch( cartsoln, on, false );
  WHILE ( ch <> esc ) AND ( x < numpumps ) DO
    BEGIN
      switch( solnswitch, on, false );
      stopwatch( 0.25 );
      switch( solnswitch, off, false );
      stopwatch( 0.25 );
      x := x + 1;
      IF keypress THEN
        ch := getchar( [ chr( 0 )..chr( 127 ) ], false, false );
    END;
  switch( solenoid, off, false );
  switch( cartsoln, off, false );
  cart.addedtoday := cart.addedtoday + x;
  goxy( 0, error_line + 2 );
  clreoln;
END;    { pump }

BEGIN   { soln_check }
  WITH cart, wrk DO
    add := ( ( cart_type = hydroponic ) AND
           ( data[ soln_chan ] < solninfo.solution ) ) OR
           ( ( cart_type = soil ) AND
           ( data[ soln_chan ] > solninfo.solution ) );
  IF add THEN
    pump( position, cart.solninfo.amount );
END;    { soln_check }


PROCEDURE tran_check( position: integer; VAR wrk: workrec );
{ resolve transpired solution overflow                                      }

BEGIN
  IF ( wrk.data[ tran_chan ] > cart.solninfo.transpired ) THEN
    BEGIN
      switch( transswitch, on, false );
      stopwatch( 0.25 );
      switch( transswitch, off, false );
    END;
END;  { tran_check }


PROCEDURE pH_check( position: integer; VAR wrk: workrec );
{ resolve pH problems                                                       }

VAR
  add_base,
  add_acid      :       boolean;

PROCEDURE pH_control( pH_switch, position: integer; bool: boolean );

VAR
  ch    :       char;

BEGIN
  goxy( 0, error_line + 2 );
  clreoln;

  prompt( 'Adding ' );                        { Show what is going on     }
  IF bool THEN prompt( 'acid ' )
  ELSE prompt( 'base ' );
  prompt( 'for position #' );
  prompt_real( position, 3, 0 );

  switch( pH_switch, on, false );             { turn on acid/base valve   }
  stopwatch( cart.pH_info.timetoadd );        { wait for specified time   }
  switch( pH_switch, off, false );            { valve off                 }
  IF keypress THEN                            { check for escape          }
    ch := getchar( [ chr(0)..chr(127) ], false, false );
  goxy( 0, error_line + 2 );
  clreoln;
END;  { pH_control }

BEGIN   { pH_check }
  add_base := wrk.data[ pH_chan ] < cart.pH_info.lower_pH;
  add_acid := wrk.data[ pH_chan ] > cart.pH_info.upper_pH;
  IF add_base THEN
    pH_control( base_switch, position, false )  { add base                  }
  ELSE IF add_acid THEN
    pH_control( acid_switch, position, true );  { add acid                  }
END;    { pH_check }


PROCEDURE lite_check;
{ if lights are off, switch co2 flow off                                    }

BEGIN
  WITH cart DO
    IF last_lite < lightinfo THEN
      BEGIN
        switch( co2off, on, false );            { toggle co2 flow off       }
        stopwatch( 0.25 );
        switch( co2off, off, false );
      END;
END; { lite_check }


PROCEDURE fatal_err( position: integer;         { the position to check.    }
                     VAR wrk : workrec );       { check these values.       }
{ check the values in the workfile record to make sure that none have       }
{ occurred outside the fatal error range.                                   }

VAR
  channum       : integer;
  error_occurred: boolean;

BEGIN
  channum := 0;
  error_occurred := false;
  REPEAT
    WITH wrk, misc.errors DO
      IF ( data[ channum ] <> missing ) AND
         ( length( labels[ channum ] ) > 0 ) THEN
           error_occurred := ( data[ channum ] < errorranges[ 1, channum ] ) OR
                             ( data[ channum ] > errorranges[ 2, channum ] );
    IF NOT error_occurred THEN
      channum := channum + 1;
  UNTIL (channum > maxchan) OR error_occurred;
  WITH misc.errors DO
    IF error_occurred THEN
      BEGIN
        errorposition := position;
        errorchan := channum;
        currerror := chanerror;
        errortime := readtime;
      END
    ELSE
      BEGIN
        currerror := noerror;
        errortime := 0;
      END;
  perform( update_parms( currvals.workvolume, mirec ) );
END;  { fatal_err }


BEGIN  { adjust }
  cart_select( position );                      { enable the position       }
  perform( load_parms( currvals.workvolume, position ) );
                                                { load cart parameters      }
  soln_check( position, wrk );                  { check solution level      }
  tran_check( position, wrk );                  { check transpired solution }
  pH_check( position, wrk );                    { check pH                  }
  lite_check;                                   { check lights on/off       }
  IF misc.errors.callok THEN
    fatal_err( position, wrk );                 { check for system errors   }
  perform( update_parms( currvals.workvolume, position ) );
                                                { update cart parameters    }
  cart_select( off_cart );                      { disable the position      }
END;    { adjust }


{-----------------------------------------------}
{ quit - handles all of the chaining and final  }
{ update for the Control program.               }
{-----------------------------------------------}

segment PROCEDURE quit( ch: char );


PROCEDURE set_passage;
```

```
{ sets up s with currvals.workvolume, ; , and %CONTROL.CODE for next program }

VAR
  s     :       string;

BEGIN
  str( currvals.workvolume, s );
  s := concat( s, ';%CONTROL.CODE' );
  setcval( s );
END;    { set_passage }


PROCEDURE examine;
{ chain to the examine program }

BEGIN
  setchain( '%EXAMINE.CODE' );
  promptat( 0, 10, 'Loading Examine Program... ' );
END;    { examine }


PROCEDURE define;
{ Chain to the define parameters program }

BEGIN
  setchain( '%DEFINE.CODE' );
  promptat( 0, 10, 'Loading Define Parameters program... ' );
END;    { define }


PROCEDURE select;
{ for selecting data from the corvus --> floppies }

BEGIN
  setchain( '%SELECT.CODE' );
  promptat( 0, 10, 'Loading select/merge program... ' );
END;    { select }


PROCEDURE setclock;
{ for setting the CCS clock }

BEGIN
  setchain( '%SETCLOCK.CODE' );
  promptat( 0, 10, 'Loading set clock program... ' );
END;    { setclock }


PROCEDURE testio;
{ for testing the hardware devices and the assembly code }

BEGIN
  setchain( '%TESTIO.CODE' );
  promptat( 0, 10, 'Loading test devices program... ' );
END;    { testio }


PROCEDURE edit_keys;
{ for editing the keyfile associated with the environmental data }

BEGIN
  setchain( '%KEYSEDIT.CODE ' );
  promptat( 0, 10, 'Loading keyfile edit program... ' );
END;    { edit_keys }


PROCEDURE edit_labels;
{ for editing the label file associated with the environmental data }
```

```
BEGIN
  setchain( '%LABEDIT.CODE' );
  promptat( 0, 10, 'Loading labels edit program... ' );
END;    { edit_labels }


PROCEDURE edit_work;

BEGIN
  setchain( '%WRKEDIT.CODE' );
  promptat( 0, 10, 'Loading workfile edit program...' );
END;    { edit_work }


BEGIN   { quit }
  {$R chainstuff}
  IF NOT modem_in_use THEN
    BEGIN
      set_passage;
      goxy( 0, 2 );
      clreosc;
      CASE ord( ch ) OF
        {O} 89:     select;
        {F} 70:     examine;
        {H} 72:     testio;
        {I} 73:     define;
        {J} 74:     edit_keys;
        {K} 75:     edit_labels;
        {L} 76:     setclock;
        {S} 83:     edit_work;
      END;  { case }
      exit( PROGRAM );
    END;        { if not modem_in_use... }
END;    { quit }


{---------------------------------------------- }
{ segment call - The segment that performs all  }
{ of the alert phone calling to lab technicians  }
{ when a measurement falls outside of the error  }
{ range set for it.                              }
{---------------------------------------------- }

segment PROCEDURE call;

VAR
  phonefile                     : interactive;
  matchset                      : setofchar;
  time,
  sec,
  last,
  len,
  i,
  count                         : integer;
  phonelist                     : ARRAY[ 1..4 ] OF string;
  response_received             : boolean;
  dow                           : integer;      { day of the week }
  s                             : string;

FUNCTION day_of_week( date: ucsd_date ): integer;
{ calculate the day of the week using Zeller's congruence                }

CONST
  century       =       19;

BEGIN
```

```
  IF date.mo < 3 THEN
    BEGIN
      date.mo := date.mo + 12;
      date.yr := date.yr - 1;
    END;
  date.mo := date.mo - 2;
  day_of_week := ( trunc( 2.6 * date.mo - 0.2 )
                   + date.day + date.yr
                   + ( date.yr DIV 4 )
                   + ( century DIV 4 ) - 2 * century ) MOD 7;
END;    { day_of_week }

PROCEDURE set_parameters;

BEGIN
  WITH misc.errors DO
    BEGIN
      IF response_received THEN
        currerror := noerror;
      IF response_received THEN
        errortime := 0
      ELSE
        errortime := errortime + errorinterval;
      perform( update_parms( currvals.workvolume, mirec ) );
    END;        { with misc.errors... }
END;  { set_parameters }


PROCEDURE check_escape;

VAR
  ch    :       char;

BEGIN
  IF keypress THEN
    BEGIN
      ch := getchar( [ chr( 0 )..chr( 127 ) ], false, false );
      IF ch = esc THEN
        BEGIN
          set_parameters;
          exit( call );
        END;        { if ch =... }
    END;        { if keypress }
END;  { check_escape }


BEGIN   { call }
  time := readtime;
  dow := day_of_week( date );
  response_received := false;
  FOR i := 1 TO 4 DO
    phonelist[ i ] := '';
  {$I-}
  reset( phonefile,'PHNUMBERS.TEXT' );
  {$I+}
  IF ioresult <> 0  THEN
    BEGIN
      set_parameters;
      exit( call );
    END;
  gotoxy( 0, error_line );
  clreosc;
  promptln( 'Error alert procedure' );
  promptln( '<esc> to abort Error alert' );
  IF ( dow IN [ 1..5 ] ) AND ( ( time >= 0800 ) AND ( time <= 1700 ) ) THEN
    matchset := [ '1','4','5','*' ]
  ELSE
    IF ( dow IN [ 1..5 ] ) AND ( ( time < 0800 ) OR ( time > 1700 ) ) THEN
```

```
      matchset := [ '2','4','6','*' ]
    ELSE
      matchset := [ '3','5','6','*' ];
  prompt( 'Searching phone number file...' );
  last := 0;
  REPEAT
    readln( phonefile,s );
    IF length( s ) > 0 THEN
      IF ( s[ 1 ] IN matchset ) AND ( last <= 4 ) THEN
        BEGIN
          last := last + 1;
          IF length( s ) < 13 THEN
            len := length( s )
          ELSE
            len := 13;
          phonelist[ last ] := copy( s, 3, ( len - 2 ) );
        END;
    check_escape;
  UNTIL eof( phonefile ) OR ( last >= 4 );
  close( phonefile );
  count := 1;
  REPEAT
    i := 1;
    gotoxy( 0, error_line );
    clreosc;
    initcomm( 2 );
    REPEAT
      IF length( phonelist[ i ] ) > 0 THEN
        BEGIN
          check_escape;
          setmode( originate );

          promptln( phonelist[ i ] );
          prompt( 'Dialing, pass #' );
          prompt_real( count, 1, 0 );
          promptln( '' );
          dial( phonelist[ i ] );

          sec := 0;
          gotoxy( 0, error_line );
          clreoln;
          promptln( 'Waiting for response' );
          REPEAT                      { wait about 60 secs???? }
            sec := sec + 1;
            response_received := carrier;
            check_escape;
          UNTIL ( sec > 7000 ) OR response_received;
          unconnect;
        END;
      gotoxy( 0, error_line );
      clreosc;
      IF response_received THEN
        prompt( 'Response received' )
      ELSE
        prompt( 'No response received' );
      i := i + 1;
      check_escape;
    UNTIL ( i > 4 ) OR response_received;
    count := count + 1;
    check_escape;
  UNTIL ( count > 3 ) OR response_received;
  set_parameters;
END;  { call }
```

AUXILIARY PROGRAMS

The three auxiliary programs are the Examine program, the Define Parameters program and the Select/Merge program. (Note - these programs are being revised. There may be some inconsistencies between these programs and the CONTROL program.)

All of the following programs include the following files:

a) CONSTS.TEXT	(See page 2-91)
b) TYPES.TEXT	(See page 2-95)
c) PARAMTYPE.TEXT	(See page 2-97)
d) VARS.TEXT	(See page 2-100)

EXAMINE (examine.code). This program allows display of the data that is stored in the workfile. See the User's guide, page 24, for a description. The text files used in the Examine program are:

a) EXAMINE.TEXT	(Host file, page 2-101)
b) ODFBLOCK.TEXT	(See page 2-108)
c) WORKLIST.TEXT	(See page 2-110)
d) PLOTBLOCK.TEXT	(See page 2-113)
e) RANGEBLOCK.TEXT	(See page 2-117)
f) FOTOBLOCK.TEXT	(See page 2-124)

DEFINE PARAMETERS (define.code). Edit and display the current parameters used by CONTROL with this program. See the User's guide, page 8 for a description. Note that the define program has its own regular unit, definestuff. Use the following text files to compile Define.code:

a) DEFINE.TEXT	(Host file, page 2-127)
b) DEFSTUFF.TEXT	(Unit, compile separately, page 2-132)
c) DEFINE1.TEXT	(See page 2-140)
d) DEFINE2.TEXT	(See page 2-150)

SELECT/MERGE (select.code). Use this program to move workfile records from the Corvus hard disk drive to floppy disk. See the User's guide, page 29 for a description. The following files comprise the Select program:

a) SELECT.TEXT	(Host file, page 2-159)
b) CONSTS.TEXT	(See page 2-91)
c) TYPES.TEXT	(See page 2-95)
d) PARAMTYPE.TEXT	(See page 2-97)
e) SELECT.INC.TEXT	(See page 2-167)

```
{______________________________________________________________
|                                                              |
| consts: an include file for the Control program. 24 Jun 85  |
| These are all the constants used in the Control program.    |
|_____________________________________________________________}


{_____________________________________________________________________________
|                                                                             |
|                        MISCELLANEOUS CONSTANTS                              |
|____________________________________________________________________________}


maxwrkfield          =      35;      { max # of fields (0..35), Statpro }

maxposition          =      17;      { max # of positions (0..17) }

maxcart              =      99;      { max # of carts (0..99) }

maxchan              =      18;      { max # of channels (0..18) }

maxswitch            =      15;      { number of switches + 1 }

thedate              =      -21252;  { Pascal 1.2 system date dec address }

samplecount          =      20;      { mean of 20 samples => reading }

echo                 =      true;    { for kybdstuff }

noecho               =      false;

noerror              =      0;       {no error currently}

loginerror           =      1;       {no response to login request}

chanerror            =      2;       {error in reading a channel}

lagtime              =      20.0;    { wait time after co2 inject }

period               =      5; { measurement period length in minutes }

maxrec               =      maxint;   { max number of records in workfil }

maxstatrec           =      20521;   { maximum statistics record number }
                                     { 60 periods worth, about 30 days }
swrec                =      18;      { switches parameter record number }

chrec                =      19;      { channels parameter record number }

errec                =      20;      { errors parameter record number }

mirec                =      21;      { misc parameter record number }

curec                =      22;      { current values param record number }

crit_light           =      0.5;     { lights off value, for transition }

nitrogen             =      -2;      { code for nitrogen standard }

carbon_dioxide       =      -1;      { code for co2 standard }

maxreal              =      1E32;    { very large real number }

minreal              =      -1E32;   { small real number }
```

```
statgroup           =      342;  { # of records of stats for one period }

{_______________________________________________________________________
|                                                                       |
|                        SWITCHES CONSTANTS                             |
|_______________________________________________________________________}

vent                =      0;  { vent to the outside }
airswitch           =      1;  { outside air }
transoff            =      2;  { turn off transpired solution dump }
transswitch         =      3;  { transpired soln drain }
co2on               =      4;  { turn co2flow back on }
co2off              =      5;  { turn co2flow off }
elecswitch          =      6;  { electrical conductivity }
n2gas               =      7;  { std nitrogen sample }
acidswitch          =      8;  { add acid }
airpump             =      9;  { draw air from the chambers }
squirt              =      10; { co2 inject }
cartsoln            =      11; { soln on cart }
solenoid            =      12; { needed for soln pump }
solnswitch          =      13; { add solution }
co2stdgas           =      14; { std co2 sample }
baseswitch          =      15; { add base }


{_______________________________________________________________________
|                                                                       |
|                        CHANNELS CONSTANTS                             |
|_______________________________________________________________________}

phchan              =      5;      { pH data channel }
lightchan           =      6;      { light measurement }
solnchan            =      7;      { solution level }
transchan           =      8;      { transpired soln data }
elecchan            =      10;     { electrical conductivity }
gaschan             =      11;     { co2 concentration }
h2ochan             =      12;     { water vapor }
massflow            =      13;     { co2 flow control reading }
```

```
tottoday                =       16;     { solution added data }

transperplant           =       17;     { transpired solution data per plant }
{ note  this is total transpired solution during the current period, not just
  the measured transpired solution }

photoperplant           =       18;     { photosynth rate estimate per plant }
{ these last two could easily be eliminated - the calculations for them
  could be performed by any statistical or post-processing program.  If
  any channels are ever needed for something important, get rid of these
  first. }

{_______________________________________________________________________________
|                                                                              |
|                         WORKFILE FIELDS CONSTANTS                            |
|______________________________________________________________________________}

{ channels 0-18 will be stored in the workfile datarec, fields 0-18 }

poscartfield            =       19;     { format, pos.cart }

datefield               =       20;     { todays date, format -> MM.DDYY }

timefield               =       21;     { time of measurement, HH.MMSS }

co2timefield            =       22;     { time of co2 measurement }

n2field                 =       23;     { N2 std gas measurement }

co2field                =       24;     { CO2 standard gas measurements }

stdtimefield            =       25;     { field for time of std. gas measure }

elapsedfield            =       26;     { field for the elapsed time }

plntfield               =       27;     { field for the number of plants }

t_rate_chan             =       28;     { transpired solution rate - calc. }


{_______________________________________________________________________________
|                                                                              |
|                    STATISTICS WORKFILE FIELD CONSTANTS                       |
|______________________________________________________________________________}


chanfield               =       1;      { channel of measurement }

firstdatefield          =       2;      { first date in the period }

firsttimefield          =       3;      { first time in this period }

lastdatefield           =       20;     { last date in this period }

lasttimefield           =       21;     { last time in this period }

nfield                  =       6;      { number of observations }

sumy                    =       7;      { sum of the measured values }

sumysqr                 =       8;      { sum of squared measureds }

maxfield                =       9;      { max measured value }

minfield                =       10;     { min measured value }

periodfield             =       11;     { period code number }
```

```
sumx                =     12;     { sum of elapsed times for linear reg }

sumxsqr             =     13;     { sum of squared elapseds }

sumxy               =     14;     { sum of value * elapsed }

meanfield           =     15;     { calculated mean of measured value }

sdfield             =     16;     { calculated std deviation }

cvfield             =     17;     { calculated coeff of variation }

slopefield          =     18;     { slope of line - linear reg }

{ 19, 20, and 21 are already used, see constants for regular workfile }

yintfield           =     22;     { y intercept of line - linear reg }

rfield              =     23;     { how well line fits data }

errfield            =     24;     { standard error of line }


{_______________________________________________________________________
|                                                                       |
|                          CLOCK CONSTANTS                              |
|_______________________________________________________________________}

slot                =     6;            {clock slot number}

start               =     0;            {clock digit latch bytes}

sec                 =     48;

min                 =     50;

hrs                 =     52;

dayofweek           =     54;

dayy                =     55;

mon                 =     57;

yrs                 =     59;
```

```
{_______________________________________________________________________________
|                                                                               |
|  types: an include file for the Control program. 26 March 1985                |
|  All type declarations for the Control program.                               |
|_______________________________________________________________________________}


{_______________________________________________________________________________
|                                                                               |
|  This is the format of the Statpro workfile.                                  |
|  Workrec is set up as 0..35 to make it easier to work with the existing       |
|  channel setup, instead of the usual 1..36.                                   |
|_______________________________________________________________________________}
workrec                =      record
                                data          : array[0..maxwrkfield]
                                                                of real;
                                dflag         : char;
                              end;

{ keyfile format }
keyrec                 =      record
                                title         : string[40];
                                rtotal,
                                fieldnum,
                                fnum          : integer;
                              end;

string9                =      string[9];

{ label file format }
labelrec               =      record
                                clabel        : string9;
                              end;

{ variant record trick for syscom date }
setdate                =      record
                                case boolean of
                                  false       :  (dateloc: integer);
                                  true        :  (dateptr: ^daterec);
                              end;

threechar              =      packed array[0..2] of char;

{ gas measurement storage record }
gastype                =      record
                                taken         :  boolean;
                                time,
                                photodata,
                                h2odata,
                                co2data       :  real;
                              end;

{ conditions of the air system }
airtype                =      (fill,purge,gasmeas);

{ periods which occur in 24 hours }
{ tran_to_day --> next period is day, tran_to_nite --> next period is nite }
period_type            =      (tran_to_nite, night, tran_to_day, day);

{ note pad for jotting down a time when something occurred.  Used often }
time_type              =      record
                                lastsecs,
                                lasttime      :  integer;
                                lastdate      :  daterec;
                              end;
```

```
fourstr                 =       string[4];

{ buffer used for reading menus from disk }
buftype                 =       packed array[1..1024] of char;

{ operating system info record }
syscomrecord            =       packed array[0..511] of char;

{ The following are disk directory types .......}
vid                     =       string[7];

fid                     =       string[15];

filekind                =       (volume,xdisk,code,text,info,data,
                                 graf,foto,securedir);

direntry                =       record
                                  dfirstblk     :       integer;
                                  dlastblk      :       integer;
                                  case dfkind   : filekind OF
                                    securedir, volume:
                                           (dvid      : vid;
                                            deovblk   : integer;
                                            dnumfiles: integer;
                                            dloadtime: integer;
                                            dlastboot: daterec);
                                    xdisk, code, text, info, data, graf, foto:
                                           (dfid      : fid;
                                            dlastbyte: 1..512;
                                            daccess   : daterec)
                                end;

directory               =       packed array[0..77] of direntry;

dirptr                  =       ^directory;

dir19                   =       packed array[0..19] of direntry;

{ the following are used for the clock stuff ..}
byte                    =       0..255;

trixarray               =       packed array[0..1] of byte;

fyletype                =       (work,stat);

setofvolumes            =       set of 4..12;

trans_array             =       array[1..6] of
                                  record
                                    value,
                                    etime    : real;
                                    abs_time : time_type;
                                  end;
```

```
{____________________________________________________________________
|                                                                    |
| Parameter file types - an include file for the Control program. 28 Mar 85
| The parameter file is a variant record file, each record has a constant
| record number.  See the consts file.                               |
|____________________________________________________________________|


 ____________________________________________________________________
|                                                                    |
| Paramfile will be declared a file of paramrec.                     |
|                                                                    |
|   Records             Contents                                     |
|                                                                    |
|   0-maxposition        cartp record for each position.             |
|                        these can be loaded from separate cart files which
|                        are of type cart. Not permanent. The proper record
|                        is loaded only when needed.                 |
|                                                                    |
|   maxposition+1        switp record - permanently loaded during run.
|                                                                    |
|   maxposition+2        chanp record - also permanent.              |
|                                                                    |
|   maxposition+3        errp - permanent.                           |
|                                                                    |
|   maxposition+4        miscp - permanent.                          |
|                                                                    |
|   maxposition+5        currp - permanent.                          |
|____________________________________________________________________}


{ Part of the channels record declaration }
channelrec              =       packed record
                                  scale         :       voltrange;
                                  name          :       string[20];
                                  { graphics scaling factors }
                                  maxscale      :       real;
                                  minscale      :       real;
                                end;  { 16 words }

{ Channels record declaration. 608 bytes or 304 words. Largest record }
chaninfo                =       array[0..maxchan] of channelrec;

{ Switches record }
switchrec               =       array[0..maxswitch] of string[20];

{ pH information, part of the positions record }
phrec                   =       packed record
                                  upperph:      real;
                                  lowerph:      real;     { in pH units }
                                  timetoadd:    integer;  { time in seconds }
                                end; { 5 words }

{ Solution information, part of the positions record }
solnrec                 =       packed record
                                  transpired :  real;{ dump transpired }
                                  solution   :  real;{ add solution }
                                  amount     :  integer;
                                  { number of pulses of solution to add }
                                end; { 5 words }

{ CO2 information, part of the positions record }
cartco2                 =       record
                                  optimum    :  real;
                                  span       :  real;   { in ppm }
                                end;  { 4 words }

{ conversion parameters for each cart, 4th order eqns, part of }
{ the positions record, 190 words }
equations               =       packed array[1..5,0..maxchan] of real;

{ the type of cart, part of the positions record }
kindcart                =       (hydroponic,soil,extra1,extra2);

{ Positions record.  Information for each cart. 254 words }
cartinfo                =       packed record
                                  cartnum       : integer;
                                  hoodvol       : real; { hood volume, liters }
                                  carttype      : kindcart;
                                  aironline     : boolean;
                                  online        : boolean;
                                  phinfo        : phrec;
                                  solninfo      : solnrec;
                                  co2info       : cartco2;
                                  conversions   : equations;

                                  { boundary between lights on/off,
                                    units: micromoles/sec/square meters }
                                  lightinfo     : real;

                                  newair        : boolean;
                                  num_plants    : integer;{ current # plants }
                                  currrate      : real;   { current co2 flow }
                                  lastco2       : real;   { last co2 measure }

                                  { transpired solution constant to keep track
                                    of total transpired solution in cart }
                                  tottranspired : real;

                                  { storage to calculate transpired solution
                                    rate }
                                  trans_rate    : trans_array;
                                end;  { cart }


{ Errors record }
errorrec                =       packed record
                                  errorranges   :array[1..2,0..maxchan]
                                                                   of real;
                                  errortime     :       integer;
                                  errorinterval :       integer;
                                  currerror     :       integer;
                                  errorchan     :       integer;
                                  errorposition :       integer;
                                  callok        :       boolean;
                                end;

{ Miscellaneous record }
miscrec                 =       record
                                  airsamprate   :  integer;
                                  samprate      :  integer;
                                  { transition period len, mins }
                                  transperiod   :  integer;
                                  lightcart     :  integer;
                                  missingflag   :  real;
                                  flowrate      :  real; { liters/second }
                                  relperiod     :  integer; {for stat display }
                                  pos           :  integer;
                                  chan          :  integer;
                                  skip          :  integer;
                                  { number of points in transpiration rate
                                    calculation }
                                  numpoints     :  integer;
                                end;

{ Current values record }
currrec                 =       record

                                  { switchstate }
                                  switchvals    : packed array[0..maxswitch] of
                                                                       boolean;
                                  { soln added }
                                  addedtoday    : array[0..maxposition] of
                                                                       integer;

                                  { last light reading }
                                  lightdata     : real;

                                  { last set of co2 measurements }
                                  { -2 for co2 std, -1 for n2 std }
                                  gasarray      : array[-2..maxposition] of
                                                                       gastype;
                                  ratetimes     : array[0..maxposition] of
                                                                      timetype;

                                  { next selection in the air loop }
                                  aircondition  : airtype;
                                  gascart       : integer;

                                  airmeasure    : boolean;
                                  regmeasure    : boolean;
                                  answerok      : boolean;
                                  statfull      : boolean;
                                  modemonline   : boolean;
                                  beep          : boolean;
                                  full          : boolean;

                                  screennum     : integer;
                                  statvolume    : integer;
                                  workvolume    : integer;
                                  airstring     : string[30];
                                  currperiod    : periodtype;
                                  lastperiod    : periodtype;
                                  skipcount     : integer;
                                end;


parmkind                =       (chanp, cartp, switp, errp, miscp, currp);

{ This is the variant record structure of the file itself }
paramrec                =       record case parmkind of
                                  cartp :       (cartfile      :  cartinfo);
                                  switp :       (switchfile    :  switchrec);
                                  chanp :       (chanfile      :  chaninfo);
                                  errp  :       (errorfile     :  errorrec);
                                  miscp :       (miscellaneous :  miscrec);
                                  currp :       (currfile      :  currrec);
                                end;
```

```
{______________________________________________________________________________
|                                                                             |
|  vars -- include file for the Control program,  11 September 84             |
|  Global variable declarations.                                              |
|_____________________________________________________________________________}

   ch                    :       char;

   recnum,                       { absolute record number }
   statrecnum,                   { next statistics record }
   baseptr,                      { for statistics summing }
   time,                         { current time }
   lastsec,                      { seconds last time clock was looked at }
   samptime,                     { time, hrs and mins, of next measurement }
   sampseconds,                  { time, seconds, of next measurement }
   airtime,                      { time, hrs and mins, of next air operation }
   airseconds,                   { time, seconds, of next air operation }
   transtime,                    { time transition period is ended }
   chan,
   dunit                 :       integer;        { volume numbers }

   s                     :       string;

   { parameter information }
   channels              :       chaninfo;  { channel record }
   switches              :       switchrec; { switches record }
   cart                  :       cartinfo;  { Position record }
   errors                :       errorrec;  { errors record }
   misc                  :       miscrec;   { miscellaneous record }
   currvals              :       currrec;   { current values record }
   paramfile             :       file of paramrec;

   statwork,                                     { statistics workfile }
   workfile              :       file of workrec;{ regular data workfile }
   keyfile               :       file of keyrec;
   monames               :       array[1..12] of threechar; { month names }
   vrnames               :       array[plus5v..disconnect] of string[8];
   pokedate              :       setdate;
   today                 :       daterec;

   { tables of air online and position online position-cart pairs }
   gaspos,
   poscart               :       packed array[0..maxposition] of -1..maxcart;

   { original date and time, for elapsed time calculations }
   origtime,
   origsecs              :       integer;
   origdate              :       daterec;
```

```
{$S++}
{$C version 11 Apr 1985 }

Program examine;

{-----------------------------------------------------------------------------

                        Examine Data program

                              for the

                   Environmental Protection Agency
                  Plant Exposure Chamber Experiments

              Corvallis Environmental Research Laboratory

                           by Mike McDowell

-----------------------------------------------------------------------------}


Uses
  chainstuff,
  turtlegraphics,
  {$U libwrk:kybd.orig.code}
  kybdstuff;

Const

{$I contwrk:consts.text}

Type

voltrange =  (plus5V,plus1V,plus500mV,plus100mV,
                 dual5V,dual1V,dual500mV,dual100mV,disconnect);

daterec =    packed record
                 mo:  0..12;
                 day: 0..31;
                 yr:  0..99;
             end;

{$I contwrk:types.text}

{$I contwrk:paramtype.text}

{ continuation of types.. }

  toplot              =       record
                                pos,                    { pos to plot }
                                chan  : integer;        { channel to plot }
                              end;

  screen              =       packed array[0..8191] of char;

  screenrec           =       record case boolean of
                                true: (screenloc: integer);
                                false: (screenptr: ^screen);
                              end;

  boundary            =       record
                                recnum      : integer;
                                date,
                                time,
                                elapsedtime : real;
                              end;
  recrange              =       record
                                  first,
                                  last         : boundary;
                                end;

  direction             =       (toscreen,todisk);


var
  workfile              :       file of workrec; { fields 0..35 }
  keyfile               :       file of keyrec;
  graphics              :       boolean;        { graf or text on screen }
  plotarray             :       array[1..6] of toplot;
  ch                    :       char;
  plotcount             :       integer;        { max of 6 plot pairs }
  channels              :       chaninfo;
  paramfile             :       file of paramrec;
  missingdata           :       real;           { missing flag from params }
  recs                  :       record
                                  perm,
                                  temp: recrange;
                                end;
  poscart               :       packed array[0..maxposition] of -1..99;
  return                :       string;


Procedure zerolist;
   Forward;


Function elapsedtime(nowdate: daterec; nowtime, nowsecs: integer;
                     thendate: daterec; thentime, thensecs: integer): real;
   Forward;


Function gettime(rtime: real; var time,secs: integer): boolean;
   Forward;


Function getdate(rdate: real; var date: daterec): boolean;
   Forward;


Procedure dispscreen(menunumber: integer);
   Forward;


Procedure flipscreen;
   Forward;


{$I miscwrk:odfblock.text}


{$I miscwrk:worklist.text}
```

```
{$I miscwrk:plotblock.text}


{$I miscwrk:rangeblock.text}


{$I miscwrk:fotoblock.text}


Procedure initialize;

var i: integer;

begin

  { load missing data flag value, and channel record from params }
  fillchar(channels,sizeof(channels),0);
  missingdata:=-999.0;
  {$I-}
  reset(paramfile,'%PARAMETERS');
  {$I+}
  if ioresult=0 then
    begin
      for i:=0 to maxposition do
        begin
          seek(paramfile,i);
          get(paramfile);
          if paramfile^.cartfile.online then
            poscart[i]:=paramfile^.cartfile.cartnum
          else
            poscart[i]:=-1;
        end;
      seek(paramfile,chrec);
      get(paramfile);
      channels:=paramfile^.chanfile;
      seek(paramfile,mirec);
      get(paramfile);
      missingdata:=paramfile^.miscellaneous.missingflag;
    end;
  close(paramfile,lock);

  initturtle;
  zerolist;
  graphics:=false;
  textmode;
end;  { initialize }



Procedure zerolist;

var
  i: integer;

begin
  plotcount:=0;
  for i:=1 to 6 do
    begin
      plotarray[i].pos:=trunc(missingdata);
      plotarray[i].chan:=trunc(missingdata);
    end;
end;   { zerolist }
```

```
Procedure flipscreen;

begin
  graphics:=not graphics;
  if graphics then
    grafmode
  else
    textmode;
end;  { flipscreen }



Function elapsedtime((nowdate: daterec; nowtime, nowsecs: integer;
                     thendate: daterec; thentime, thensecs: integer): real);
{ returns the elapsed time between dates in hours and fractions of hours }

var
  nowjd,
  thenjd,
  nowhours,
  nowmin,
  thenhours,
  thenmin              :       integer;

function julian(day, month: integer): integer;
{ returns the julian day of a non-leap year }

var
  i,
  totdays              :       integer;

function daysinmo(month: integer): integer;

begin
  case month of
    1,3,5,7,8,10,12 : daysinmo:=31;
    2               : daysinmo:=28;
    4,6,9,11        : daysinmo:=30;
  end;  { case }
end;   { daysinmo }

begin   { julian }
  totdays:=0;
  for i:=1 to month-1 do
    totdays:=totdays + daysinmo(i);
  julian := totdays + day;
end;  { julian }

begin  { elapsed time }
  nowhours := nowtime div 100;
  nowmin := nowtime mod 100;
  thenhours := thentime div 100;
  thenmin := thentime mod 100;
  nowjd:=julian(nowdate.day,nowdate.mo);
  thenjd:=julian(thendate.day,thendate.mo);
  elapsedtime:= ((nowdate.yr-thendate.yr) * 8760) + ((nowjd - thenjd) * 24.0) +
               ((nowhours - thenhours) + ((nowmin - thenmin)/60)) +
               ((nowsecs - thensecs)/3600);
end;  { elapsedtime }
```

```
Function gettime{(rtime: real; var time,secs: integer): boolean};
{ calculates a time from the real Statpro time format }

var
  h,m,s: integer;

begin
  gettime:=true;
  h:=trunc(rtime);
  s:=round(((rtime * 100) - trunc(rtime*100))*100);
  m:=round(((trunc(rtime*100)/100)-(trunc(trunc(rtime*100)/100)))*100);
  if (h in [0..23]) and (m in [0..59]) and (s in [0..59]) then
    begin
      time:=trunc(h*100 + m);
      secs:=s;
    end
  else
    gettime:=false;
end; { gettime }



Function getdate{(rdate: real; var date: daterec): boolean};
{ calculates a daterec from the real Statpro date format }

var
  m,d,y: integer;

begin
  getdate:=true;
  m:=trunc(rdate);
  y:=round(((rdate * 100) - trunc(rdate*100))*100);
  d:=round(((trunc(rdate*100)/100)-(trunc(trunc(rdate*100)/100)))*100);
  if (m in [1..12]) and (y in [1..100]) and (d in [1..31]) then
    with date do
      begin
        mo:=m; day:=d; yr:=y;
      end
  else
    getdate:=false;
end;   { getdate }



Procedure dispscreen{(menunumber: integer)};
{ read in and display menu: will output a textfile which has been set }
{ up in the form of a menu.  The two screen text pages are supported. }

var
  scrnfile      : file;
  buf           : ^buftype;
  s             : string;
  bytes_scanned,
  page,
  blocks        : integer;
  heap          : ^integer;
  ch            : char;



procedure getbuffer(var buf: buftype; relblock: integer; var blkcount: integer);
```

```
begin
  fillchar(buf,sizeof(buf),0);
  blkcount:=blockread(scrnfile,buf,2,relblock);
  if blkcount = 2 then
    bytes_scanned := scan(-1024,<>chr(0),buf[1024])
  else
    bytes_scanned := 0;
end;    { getbuffer }



Procedure dooutput;
{ display menu }

begin
  for page:=1 to 2 do
    begin
      getbuffer(buf^,(page * 2),blocks);
      if blocks=2 then
        unitwrite(1,buf^,sizeof(buftype)+bytes_scanned,0,0);
    end;
end;   { dooutput }




begin   { dispscreen }
  mark(heap);
  new(buf);
  str(menunumber,s);
  s:=concat('SCREENS:EXSCRN',s,'.TEXT');
  {$I-}
  reset(scrnfile,s);
  {$I+}
  if ioresult=0 then
    dooutput
  else
    promptat(10,concat('No sceen file ',s));
  close(scrnfile,lock);
  release(heap);
end;  { dispscreen }





begin  { examine }
  {$N+}
  {$R turtlegraphics}
  getcval(return);             { program to return to }
  promptat(0,concat('Initializing',eos));
  opendatafile;             { read keys, reset workfile }
  initialize;           { init variables }
  promptat(0,eos);

  repeat
    promptat(0,'Examine: ?)  S)pecify R)ecords P)lot F)lip G)raph  Q)uit ');
    write(eol);
    ch:=readachar(' ',['?','P'..'S','F','G','p'..'s','f','g'],false);
    case ch of
      '?':     begin
                 promptat(1,eos);
                 dispscreen(1);       { displays helpscreen }
               end;
      'P','p': with recs.temp do
                 plot(first.recnum,last.recnum);    { plots items in list }
      'R','r': range;           { specify record range }
      'S','s': worklist;        { work on the list }
      'F','f': flipscreen;      { switch between graphics and text }
      'G','g': fotohandle;      { save and retrieve fotos }
    end;  { case }
  until ch in ['q','Q'];
  { close files }
  promptat(1,eos);
  if length( return ) > 0 then
    begin
      promptat(10,'Returning to ');
      write(return);
      setchain(return);
    end
  else promptat(10,'Examine program ended....');
end.
```

```
{ odfblock: the open data file segment for Examine program,  8 Apr 85 }

{ the datafile will always be on the Corvus drive, volume 11: }

Segment Procedure opendatafile;

{$I-}

var
  i            :       integer;
  ch           :       char;


Procedure cannotopen;
begin
   promptat(20,concat(bell,'Error opening files, Press <esc>'));
   ch:=readachar(esc,[],false);
   promptat(20,concat('Returning to Control Program',eol));
   setchain('%control.code');
   exit(program);
end;  {cannotopen}



Procedure getkeys;
{ get last record by accessing the keyfile }

begin
  promptat(10,concat('Reading key file....',eol));
  reset(keyfile,'#11:WKEYFILE');
  if ioresult=0 then
    begin
      recs.perm.last.recnum:=keyfile^.rtotal-1;
      recs.perm.first.recnum:=1;
      close(keyfile,lock);
    end
  else cannotopen;
end;  { getkeys }



Procedure openfiles;

begin
  reset(workfile,'#11:WORKFILE');
  if ioresult<>0 then
    condition:=abort             { flag to initialize a new workfile }
  else
    getkeys;

  { load the times and dates }
  seek(workfile,recs.perm.first.recnum);
  get(workfile);
  recs.perm.first.time:=workfile^.data[timefield];
  recs.perm.first.date:=workfile^.data[datefield];
  recs.perm.first.elapsedtime:=workfile^.data[elapsedfield];
  seek(workfile,recs.perm.last.recnum);
  get(workfile);
  recs.perm.last.time:=workfile^.data[timefield];
  recs.perm.last.date:=workfile^.data[datefield];
  recs.perm.last.elapsedtime:=workfile^.data[elapsedfield];
  recs.temp:=recs.perm;
end;  { openfiles }
```

```
begin  {opendatafile}
  promptat(10,concat('Opening current Workfile... ',eos));
  openfiles;
  if condition=abort then
    cannotopen;
end;  {opendatafile}



{$I+}
```

```
{ worklist - an include file for the examine program 31 Jul 85 }

Segment Procedure worklist;
{ work on the channel/position list }

var
  ch            :       char;

procedure viewlist(dispoptions: boolean);
{ displays the list }

var
  i             :       integer;

begin
  promptat(1,eos);
  if dispoptions then
    begin
      dispscreen(8);
      for i:=0 to maxchan do
        begin
          gotoxy(58,i+3);
          write(channels[i].name:20);
        end;
      for i:=0 to maxposition do
        begin
          gotoxy(46,i+3);
          if poscart[i]>0 then write(poscart[i]:2)
          else write('--');
        end;
    end;  { if dispoptions }

  promptat(5,'      Position  Channel');
  if (plotcount > 0) then
    begin
      for i:=1 to plotcount do
        begin
          gotoxy(8,i+6);
          write('[  ]       [  ]');
          gotoxy(9,i+6);
          write(plotarray[i].pos:2);
          gotoxy(20,i+6);
          write(plotarray[i].chan:2);
        end
    end;
end;  { viewlist }

procedure sortlist;
{ sorts list by position, with missing data flags at end of list }

var
  finished      :       boolean;
  i             :       integer;

procedure interchange(var previtem,postitem: toplot);

var
  temp          :       toplot;

begin
  finished:=false;
  temp:=previtem;
  previtem:=postitem;
  postitem:=temp;
end;  { interchange }

begin  { sortlist }
  finished:=false;
  while not finished do
    begin
      finished:=true;
      for i:=1 to 5 do
        begin
          if ((plotarray[i].pos > plotarray[i+1].pos) and
             (plotarray[i+1].pos <> missingdata)) or
             ((plotarray[i].pos = missingdata) and
               (plotarray[i+1].pos <> missingdata)) then
                 interchange(plotarray[i],plotarray[i+1]);
        end;
    end;  { while }
end;  { sortlist }

procedure additems;
{ adds pairs to the list }

var
  i                      :       integer;

begin
  viewlist(true);
  promptat(20,'<esc> to quit');
  while (plotcount<6) and (condition<>abort) do
    begin
      plotcount:=succ(plotcount);
      promptat(22,concat('Enter position number [0..17]',eol));
      gotoxy(8,plotcount+6);
      write('[  ]       [  ]');
      gotoxy(9,plotcount+6);
      plotarray[plotcount].pos:=getinteger(trunc(missingdata),17,0);
      if plotarray[plotcount].pos < 0 then condition := abort;
      if condition<>abort then
        begin
          promptat(22,concat('Enter channel number [0..35]',eol));
          gotoxy(20,plotcount+6);
          plotarray[plotcount].chan:=getinteger(trunc(missingdata),35,0);
          if plotarray[plotcount].chan < 0 then condition := abort;
          if condition = abort then
            plotcount:=pred(plotcount)
        end
      else plotcount:=pred(plotcount);
    end;   { while }
  sortlist;
  viewlist(false);
end;  { additems }

procedure delitems;

var
  todel         :       packed array[1..6] of boolean;
  numtodel,
  i             :       integer;

begin
  viewlist(false);
  promptat(20,concat('<D> to delete',eol));
  for i:=1 to plotcount do
    begin
      gotoxy(0,i+6);
      ch:=readachar(' ',['D','d'],false);
      if ch in ['D','d'] then
        begin
          todel[i]:=true;
          write('*');
        end
      else
        begin
```

```
          todel[i]:=false;
          write('+');
        end;
    end;  { for i:=1 to plotcount }
  numtodel:=0;
  for i:=1 to plotcount do
    begin
      if todel[i] then
        begin
          plotarray[i].pos:=trunc(missingdata);
          plotarray[i].chan:=trunc(missingdata);
          numtodel:=succ(numtodel);
        end;
    end;  { for i:=0 to nodecount-1 }
  plotcount:=plotcount-numtodel;
  if plotcount < 0 then plotcount:=0;
  sortlist;
  viewlist(false);
end;  { delitems }

begin   { worklist }
  {$R kybdstuff}
  promptat(0,eos);
  repeat
    promptat(0,concat('Specify:  ?)  N)ew  A)dd  D)elete  V)iew <esc> ',eol));
    ch:=readachar(' ',['?','N','n','A','a','D','d','V','v'],false);
    case ch of
      '?':     begin
                 promptat(1,eos);
                 dispscreen(3);   { display the help file }
               end;
      'N','n': zerolist;        { clear the list }
      'A','a': additems;        { add list items }
      'D','d': delitems;        { delete list items }
      'V','v': viewlist(true);  { view list items }
    end;  { case }
  until condition=abort;
end;  { worklist }
```

```
{ plotblock - an include file for the Examine program, 8 Apr 85 }

Segment Procedure plot(first,last: integer);

var
  i,
  j            :       integer;

procedure drawaxes;

var
  ystr         :       string[11];
  i            :       integer;

begin
  viewport(0,279,0,191);
  pencolor(none);
  moveto(10,10);
  pencolor(white);
  moveto(279,10);  moveto(279,191);  moveto(10,191);  moveto(10,10);
  pencolor(none);
  moveto(112,0);
  wstring('Time');
  ystr:='Measurement';
  for i:=1 to 11 do
    begin
      moveto(0,145-i*8);
      wchar(ystr[i]);
    end;
  viewport(11,278,11,190);
  fillscreen(black);
end;    { drawaxes }

procedure display(work: workrec; which: integer);
{ scales and displays the value }
{ which is the item in the array }

var
  x,
  y            :       integer;

procedure putpoint(which: integer);

begin
  case which of
    1: begin { a dot }
         pencolor(white);
         move(0);
         pencolor(none);
       end;
    2: begin  { draw a little cross }
         turnto(0);
         move(1);
         pencolor(white);
         move(-2);  move(1);
         turnto(90);  move(1);  move(-2);
         pencolor(none);
       end;
    3: begin  { an x }
         turnto(45);
         pencolor(white);
         move(1);  move(-3);  move(1);
         turnto(135);  move(1);  move(-3);
         pencolor(none);
       end;
    4: begin  { empty box }
```

```
         turnto(0);
         move(1);
         pencolor(white);
         turn(135);  move(2);
         turn(90);  move(1);
         turn(90);  move(1);
         turn(90);  move(1);
         pencolor(none);
       end;
    5: begin { box }
         turnto(0);
         pencolor(white);
         move(1);
         turn(90);  move(1);
         turn(90);  move(2);
         turn(90);  move(2);
         turn(90);  move(2);
         pencolor(none);
       end;
    6: begin  { circle }
         turnto(0);
         move(1);
         pencolor(white);
         turn(45);  move(1);
         turn(45);  move(1);
         turn(45);  move(1);
         turn(45);  move(1);
         turn(45);  move(1);
         turn(45);  move(1);
         turn(45);  move(1);
         pencolor(none);
       end;
  end;
end;  { putpoint }

function xscale(xval: real): integer;
{ scale time axis }  { xval is the elapsed time of the record to plot }

begin
  with recs.temp do
    xscale:=round(11 +
       (((xval-first.elapsedtime)/(last.elapsedtime-first.elapsedtime)) * 267));
end;  { xscale }

function yscale(yval,maxval,minval: real): integer;

begin
  if maxval-minval=0 then
    yscale:=trunc(missingdata)
  else
    yscale:=round(11 + (((yval-minval)/(maxval-minval)) * 189));
end;  { yscale }

begin { display }
  if work.data[plotarray[which].chan] <> missingdata then
    begin
      pencolor(none);
      x:=xscale(work.data[elapsedfield]);
      if plotarray[which].chan in [ 0..maxchan ] then
        with channels[plotarray[which].chan] do
          y:=yscale(work.data[plotarray[which].chan],maxscale,minscale)
      else
        y:=yscale(work.data[plotarray[which].chan],10,-1);
      if y <> trunc(missingdata) then
        begin
          moveto(x,y);
          putpoint(which);
        end;
    end;   { if work.data   etc.... }
end;  { display }

Function notthere(work: workrec): boolean;

begin
  with work do
    notthere := (data[poscartfield] = missingdata) or
      (data[plotarray[j].chan] = trunc(missingdata));
end;   { notthere }


Function legalrec(work: workrec): boolean;

begin
  legalrec:=false;
  if not notthere(work) then
    if (work.data[poscartfield] <> missingdata) and
        (work.data[plotarray[j].chan] <> missingdata) then
          if trunc(work.data[poscartfield]) = plotarray[j].pos then
            legalrec:=true;
end;   { legalrec }


Procedure summary;

begin
  promptat(1,eos);
  dispscreen(7);
  with recs.temp do
    begin
      gotoxy(15,5);                     write(first.recnum:5);
      gotoxy(31,5);                     write(last.recnum:5);
      gotoxy(15,6);                     write(first.date:8:4);
      gotoxy(31,6);                     write(last.date:8:4);
      gotoxy(15,7);                     write(first.time:8:4);
      gotoxy(31,7);                     write(last.time:8:4);
    end;
  for i:=1 to plotcount do
    begin
      gotoxy(1,i+11);                    write(plotarray[i].pos:2);
      gotoxy(6,i+11);                    write(plotarray[i].chan:2);
      gotoxy(11,i+11);
      case i of
        1:  write('dot':6);
        2:  write('plus':6);
        3:  write('x':6);
        4:  write('square':6);
        5:  write('box':6);
        6:  write('circle':6);
      end;
      if plotarray[i].chan in [0..maxchan] then
        begin
          gotoxy(20,i+11);
          write(channels[plotarray[i].chan].minscale:8:4);
          gotoxy(31,i+11);
          write(channels[plotarray[i].chan].maxscale:8:4);
        end
      else
        begin
          gotoxy(20,i+11);
          write('-1.0000':8);
          gotoxy(31,i+11);
          write('10.0000':8);
        end;
    end;  { for i:=1 to plotcount }
end;  { summary }
```

```
begin  { plot }
  promptat(1,eos);
  promptat(5,'Plotting.....');
  drawaxes;
  if recs.temp.last.elapsedtime-recs.temp.first.elapsedtime=0 then
    begin
      if graphics then flipscreen;
      promptat(20,'No range of time for scaling, Push <esc> ');
      ch:=readachar(' ',[],false);
      exit(plot);
    end;
  if plotcount > 0 then
    begin
      if not graphics then flipscreen;
      seek(workfile,first);

      for i:=first to last do
        begin
          get(workfile);
          for j:=1 to plotcount do
            begin
              if keypress then
                begin
                  ch:=readachar(' ',[chr(0)..chr(127)]-contchar,false);
                  if condition=abort then exit(plot);
                end;
              if legalrec(workfile^) then
                display(workfile^,j);
            end;     { for j:=1 to plotcount }
        end;  { for i:= first to last }
      summary;

    end;  { if plotcount > 0 }
 end;  { plot }
```

```
{ rangeblock - an include file for the Examine program, 9 Apr 85 }


Segment Procedure range;
{ specify the record range to plot, pick up firsttime, lasttime }

var
  ch            :       char;
  subscript     :       integer;


Procedure move(subscript: integer);

begin
  if odd(subscript) then
    if subscript < 6 then
      gotoxy(31,(subscript div 2) + 7)
    else
      gotoxy(31,(subscript div 2) + 8)
  else
    if subscript < 6 then
      gotoxy(19,(subscript div 2) + 7)
    else
      gotoxy(19,(subscript div 2) + 8);
end;  { move }


Procedure out(subscript: integer);


begin   { out }
  with recs do
    case subscript of
      0: write(perm.first.date:8:4);      1: write(perm.last.date:8:4);
      2: write(perm.first.time:8:4);      3: write(perm.last.time:8:4);
      4: write(perm.first.recnum:8);      5: write(perm.last.recnum:8);
      6: write(temp.first.date:8:4);      7: write(temp.last.date:8:4);
      8: write(temp.first.time:8:4);      9: write(temp.last.time:8:4);
     10: write(temp.first.recnum:8);     11: write(temp.last.recnum:8);
    end;  { case }
end;   { out }



Procedure disprange;
{ displays permanent and current range }

begin
  promptat(1,eos);
  gotoxy(0,5);
  dispscreen(5);       { screen for record range display }
  for subscript:=0 to 11 do
    begin
      move(subscript);
      out(subscript);
    end;
end;   { disprange }



Procedure timeset;
{ select record range by date and time }
```

```
var
  temp          :       integer;
  key           :       real;

procedure error(errnum: integer);

var
  ch: char;

begin
  case errnum of
    1: begin
         promptat(19,concat(bell,'Error in first date in the file, <esc>'));
         promptat(20,'Select by record number only');
         ch:=readachar(esc,[],false);
       end;
    2: begin
         promptat(19,concat(bell,'Error in first time in the file, <esc>'));
         promptat(20,'Select by record number only');
         ch:=readachar(esc,[],false);
       end;
    3: begin
         promptat(20,'Error reading date, <esc>');
         ch:=readachar(esc,[],false);
       end;
    4: begin
         promptat(20,'Error reading time, <esc>');
         ch:=readachar(esc,[],false);
       end;
  end;  { case }
  exit(timeset);
end;  { error }

Function recscale(date,time: real; var calcelapsed: real): integer;
{ returns a guess at the record number with this date and time }

var
  permdate,
  guessdate     :       daterec;
  testrec,
  permtime,
  guesstime,
  permsecs,
  guesssecs     :       integer;
  totelapsed    :       real;

begin
  with recs do
    begin
      if not getdate(perm.first.date,permdate) then error(1);
      if not gettime(perm.first.time,permtime,permsecs) then error(2);
      if not getdate(date,guessdate) then error(3);
      if not gettime(time,guesstime,guesssecs) then error(4);
      calcelapsed:=elapsedtime(guessdate,guesstime,guesssecs,
                               permdate,permtime,permsecs);
      totelapsed:=perm.last.elapsedtime - perm.first.elapsedtime;
      with perm do
        testrec:=round((last.recnum-first.recnum) * (calcelapsed/totelapsed));
      if testrec>perm.last.recnum then testrec:=perm.last.recnum
      else if testrec<perm.first.recnum then testrec:=perm.first.recnum;
    end;
  recscale:=testrec;
end;  { recscale }



Function getrec(start: boolean; guess: integer; searchkey: real): integer;
```

```
{ start indicates search for first record (=true), or last (=false) }
{ searchkey = the elapsed time value being looked for }

var
  temp          :       integer;

procedure lookforwards;

begin
  while (workfile^.data[elapsedfield] < searchkey) do
    begin
      get(workfile);
      guess:=succ(guess);
      if guess > recs.perm.last.recnum then
        searchkey:=workfile^.data[elapsedfield] - 1;
    end;
end;  { lookforwards }

procedure lookbackwards;

begin
  while (workfile^.data[elapsedfield] > searchkey) do
    begin
      guess:=pred(guess);
      if guess < recs.perm.first.recnum then
        begin
          searchkey:=workfile^.data[elapsedfield] + 1;
          guess:= recs.perm.first.recnum;
        end
      else
        begin
          seek(workfile,guess);
          get(workfile);
        end;
    end;
end;   { lookbackwards }

begin  { getrec }
  seek(workfile,guess);
  get(workfile);

  if start then { begin search for beginning }
    begin
      lookforwards;
      lookbackwards;
    end

  else { begin search for ending }
    begin
      lookbackwards;
      lookforwards;
    end;
  getrec:=guess;
end;   { getrec }



begin  { timeset }

  disprange;
  subscript:=6;
  repeat
    move(subscript);
    with recs.temp do
      case subscript of
        6: first.date:=getfpnum(first.date,last.date,1.01);
        7: last.date:=getfpnum(last.date,12.3199,first.date);
        8: first.time:=getfpnum(first.time,23.5959,0);
        9: last.time:=getfpnum(last.time,23.5959,0);
      end;  { case }
    if condition=next then subscript:=succ(subscript)
    else if condition=previous then subscript:=pred(subscript);
    if subscript>9 then subscript:=6
    else if subscript<6 then subscript:=9;
    move(subscript);
    out(subscript);
  until condition=abort;
  temp:=recscale(recs.temp.first.date,recs.temp.first.time,key);
  recs.temp.first.recnum:=getrec(true,temp,key);
  temp:=recscale(recs.temp.last.date,recs.temp.last.time,key);
  recs.temp.last.recnum:=getrec(false,temp,key);
  seek(workfile,recs.temp.first.recnum);
  get(workfile);
  with recs.temp.first do
    begin
      date:=workfile^.data[datefield];
      time:=workfile^.data[timefield];
      elapsedtime:=workfile^.data[elapsedfield];
    end;
  seek(workfile,recs.temp.last.recnum);
  get(workfile);
  with recs.temp.last do
    begin
      date:=workfile^.data[datefield];
      time:=workfile^.data[timefield];
      elapsedtime:=workfile^.data[elapsedfield];
    end;
end;  { timeset }



Procedure recset;
{ select record range by record number }

begin
  disprange;
  subscript:=10;
  repeat
    move(subscript);
    if subscript=10 then
     with recs.temp do
      first.recnum:=getinteger(first.recnum,last.recnum,recs.perm.first.recnum)
    else
     with recs.temp do
      last.recnum:=getinteger(last.recnum,recs.perm.last.recnum,first.recnum);
    move(subscript);
    out(subscript);
    if condition=next then subscript:=succ(subscript)
    else if condition = previous then subscript:=pred(subscript);
    if subscript>11 then subscript:=10
    else if subscript<10 then subscript:=11;
  until condition=abort;
  with recs.temp do
    begin
      seek(workfile,first.recnum);
      get(workfile);
      with first do
        begin
          time:=workfile^.data[timefield];
          date:=workfile^.data[datefield];
          elapsedtime:=workfile^.data[elapsedfield];
        end;
      seek(workfile,last.recnum);
      get(workfile);
      with last do
        begin
          time:=workfile^.data[timefield];
          date:=workfile^.data[datefield];
          elapsedtime:=workfile^.data[elapsedfield];
        end;
    end;  { with recs.temp }
end;  { recset }



Procedure scaleset;
{ select y axis scaling range from parameter file }

var
  flip     :    boolean;

procedure update;

begin
  {$I-}
  reset(paramfile,'%PARAMETERS');
  {$I+}
  if ioresult = 0 then
    begin
      seek(paramfile,chrec);
      get(paramfile);
      paramfile^.chanfile:=channels;
      seek(paramfile,chrec);
      put(paramfile);
      close(paramfile,lock);
    end;
end;   { update }

procedure move(subscript: integer);
begin
  if not flip then
    if odd(subscript) then
      gotoxy(31,(subscript div 2) + 6)
    else gotoxy(19,(subscript div 2) + 6)
  else
    if odd(subscript) then
      gotoxy(31,((subscript - 20 ) div 2) + 6)
    else
      gotoxy(19,((subscript - 20 ) div 2) + 6);
end;  { move }

procedure out(subscript: integer);
begin
  if subscript=38 then
    write('        ')
  else if subscript=39 then
    write('        ')
  else
    if odd(subscript) then
      write(channels[subscript div 2].maxscale:8:2)
    else write(channels[subscript div 2].minscale:8:2);
end;  { out }

procedure dispvals;

var
  subscript: integer;

begin
  if not flip then
    begin
```

```
   for subscript:=0 to 9 do
    begin
     gotoxy(1,subscript+6);
     write(subscript:2);
     gotoxy(6,subscript+6);
     with channels[subscript] do
      if length(name)>9 then
       write(copy(name,1,9):9)
      else
       write(name:9);
    end;
   for subscript:=0 to 19 do
    begin
     move(subscript);
     out(subscript);
    end;
  end
 else
  begin
   for subscript:=10 to 19 do
    begin
     gotoxy(1,subscript-4);
     if subscript=19 then write('  ')
     else write(subscript:2);
     gotoxy(6,subscript-4);
     if subscript=19 then write(' ':9)
     else with channels[subscript] do
      if length(name)>9 then
       write(copy(name,1,9):9)
      else
       write(name:9);
    end;
   for subscript:=20 to 39 do
    begin
     move(subscript);
     out(subscript);
    end;  { for subscript = 20... }
  end;  { else }
end; { dispvals }

procedure flipscreen;
begin
 flip:=not flip;
 dispvals;
end;  { flipscreen }

procedure changevals;
begin
 subscript:=0;
 promptat(21,'Enter maximum or minimum scale value');
 repeat
  move(subscript);
  with channels[subscript div 2] do
   if odd(subscript) then
    maxscale:=getfpnum(maxscale,1E32,minscale)
   else
    minscale:=getfpnum(minscale,maxscale,-1E32);
  move(subscript);
  out(subscript);
  if condition=next then subscript:=subscript+1
  else if condition=previous then subscript:=subscript-1;
  if subscript> 37 then subscript:=0
  else if subscript<0 then subscript:= 37;
  if ((subscript>19) and (not flip)) or ((subscript<20) and flip) then
   flipscreen;
 until condition=abort;
end;
```

```
begin  { scaleset }
  flip:=false;
  promptat(1,eos);
  gotoxy(0,2);
  dispscreen(6);
  promptat(17,'<-/-> move; <esc> return to Range menu');
  dispvals;
  changevals;
  update; { the parameters file }
end;   { scaleset }



begin  { range }
  {$R kybdstuff}
  repeat
    promptat(0,concat('Range:  ?)  T)ime  N)umber  S)cale <esc> ',eol));
    ch:=readachar(' ',['?','T','t','N','n','S','s'],false);
    case ch of
      '?':     begin
                 promptat(1,eos);
                 dispscreen(2);     { help screen 2 }
               end;
      'T','t': timeset;          { specify by date and time }
      'N','n': recset;           { specify by record number }
      'S','s': scaleset;         { specify scaling ranges for y axis }
    end;  { case }
  until condition=abort;
end;  { range }
```

```
{ fotoblock - an include file for the examine program, 9 Apr 85 }

Segment Procedure fotohandle;


var
  title         :       string;
  ch            :       char;
  screenfile    :       file;
  graph         :       interactive;
  pokescreen    :       screenrec;




Procedure screendump;

var
  x,
  y,
  pinnum      :       integer;

procedure initgraph;

var
  i     :       integer;

begin
  promptat(5,'Make sure Epson printer is on-line:');
  write(' <ret> or <esc> to abort ',eol);
  ch := readachar(' ',[],false);
  if condition = abort then exit(screendump);
  rewrite(graph,'printer:');
  promptat(6,'Enter a title for the graph:');
  promptat(7,'[');
  for i := 1 to 40 do
    write('.');
  write(']');
  gotoxy(1,7);
  getstring('',title,[chr(32)..chr(126)],40);
  write(graph,chr(27),'A',chr(44));
  write(graph,chr(10),chr(10),chr(10),chr(10)); {goto mid-page}
  write(graph,chr(27),'U',chr(1)); {unidirectional printing}
  write(graph,chr(27),'A',chr(6)); {row will be six dots high}
end;  {initgraph}

procedure dot(pinnum: integer); {prints printer equivalent of a pixel}
begin
  write(graph,chr(pinnum),chr(pinnum),chr(pinnum));
end;        { dot }

procedure finigraph;
begin
  write(graph,chr(27),'A',chr(12)); {return to normal spacing}
  write(graph,chr(27),'U',chr(0)); {unidirectional printing off}
  write(graph,chr(10),chr(10));

  {center and write title}
  writeln(graph,' ':(40 - (length(title) div 2)),title);
  write(graph,chr(12)); {form feed}
  close(graph);
end;

begin  {screendump}
  initgraph;
  y:=191;
  repeat
    write(graph,' ',chr(27),'L',chr(72),chr(3)); {initialize super hi-res}
    for x:=0 to 279 do                           {mode for 80 columns}
      begin
        if keypress then
          begin
            ch := readachar(' ',[chr(0)..chr(127)],false);
            if condition = abort then
              begin
                close(graph);
                exit(screendump);
              end;
          end;
        pinnum:=0;
        if screenbit(x,y) then pinnum:=pinnum+48;
        if screenbit(x,y-1) then pinnum:=pinnum+12;
        if screenbit(x,y-2) then pinnum:=pinnum+3;
        dot(pinnum);
      end;
    y:=y-3;
    writeln(graph);  {line-feed}
  until y<=0;
  finigraph;
end;  {screendump}



Procedure scrndsk(whichway: direction);
{ the screen file should already be opened. Read or write depending }
{ on whichway. }

var
  i     :       integer;

begin
  pokescreen.screenloc:=8192;
  case whichway of
    todisk: i := blockwrite(screenfile,pokescreen.screenptr^,16,0);
    toscreen: i := blockread(screenfile,pokescreen.screenptr^,16,0);
  end; {case}
end; {scrndsk}



Procedure openfile(whichway: direction);
{ open the screenfile }

var
  scrnname      :       string;
  ch            :       char;
  iocode        :       integer;

begin
  promptat(5,'Enter the file name for the screen file: ');
  write('[.......................]');
  gotoxy(42,5);
  getstring('',scrnname,[chr(32)..chr(126)],23);
  if length(scrnname) = 0 then
    exit(fotohandle);
  {$I-}
  case whichway of
    todisk      :       rewrite(screenfile,scrnname);
    toscreen    :       reset(screenfile,scrnname);
  end;
  {$I+}
  iocode := ioresult;
  if iocode <> 0 then
    begin
      promptat(10,'I/O error # ');
      write(chr(7),iocode,'; press <ret>');
      ch := readachar(' ',[],false);
      exit(fotohandle);
    end;
end;    { openfile }



  Procedure save;

  begin
    openfiles(todisk);
    promptat(20,'Saving..... ');
    scrndsk(todisk);
    close(screenfile,lock);
  end;  {savcheck}




  Procedure retrieve;

  begin
    openfile(toscreen);
    promptat(20,'Retrieving..... ');
    scrndsk(toscreen);
    close(screenfile,lock);
  end;  { retrieve }





 begin  { fotohandle }
   promptat(0,eos);
   repeat
     promptat(0,
             'Graph:    ?)  D)ump  S)ave to disk  R)etrieve from disk <esc> ');
     write(eol);
     ch := readachar(esc,['D','d','S','s','R','r','?'],false);
     promptat(1,eos);
     case ch of
       'D','d':  screendump;
       'S','s':  save;
       'R','r':  retrieve;
       '?'    :  dispscreen(4);
     end;  {case}
   until condition = abort;
 end;  {fotohandle}
```

```
PROGRAM define;
{---------------------------------------------}
{  Define program:                            }
{  A program which allows editing of the      }
{  Control program's parameter file.          }
{                                             }
{  Version 11 Oct 85    1600          -mkm-   }
{---------------------------------------------}

USES
  chainstuff,
  {$U ctrlstuff.code} control_stuff,
  {$U defstuff.code}  define_stuff;


VAR
  quit,
  last_char,
  ctrla,
  ctrlb          :      char;
  to_program     :      string;
  pos_edit       :      pos_edit_type;
  eqn_edit       :      eqn_edit_type;
  on_line        :      PACKED ARRAY[ 1..maxposition ] OF boolean;
  vol            :      integer;


{$I DEFINE1.TEXT}
{$I DEFINE2.TEXT}


segment PROCEDURE initialize( VAR last_char: char );

VAR
  i     :       integer;

PROCEDURE get_calling_info( VAR vol_num: integer; VAR prog_name: path_name );
{ checks cval for message.  If there, It returns the vol_num and program }
{ to return to.                                                          }
{ message should look like: 11;%CONTROL.CODE                             }

CONST
  delim          =      ';';    { delimiter within the message }

VAR
  message        :      string;
  position       :      integer;

BEGIN
  get_cval( message );
  set_cval( '' );
  IF length( message ) > 0 THEN
    BEGIN
      position := pos( delim, message );
      prog_name := copy( message, 1, position - 1 );    { read volume }
      vol_num := trunc( fpnum( prog_name ) );
      prog_name := copy( message, position + 1,
                                    ( length( message ) - position ) );
    END;        { if length( message )... }
END;    { get_calling_info }

PROCEDURE get_volume( VAR vol: integer; VAR last_char: char );

BEGIN   { get_volume }
  vol := 5;
  goxy( 0, 10 );
  prompt( 'Enter volume for parameter file: [ 5]' );
  goxy( 0, 20 );
  prompt( 'Enter 4, 5, 9-12' );
  REPEAT
    goxy( 34, 10 );
    vol := get_integer( vol, 12, 4, last_char );
  UNTIL ( vol IN [ 4, 5, 9..12 ] )  OR ( last_char = esc );
END;    { get_volume }

FUNCTION init_parms( vol: integer ): error_type;
{ initialize a new paramter file on this volume                            }

VAR
  s             :      string;
  i,
  iocode        :      integer;

PROCEDURE init_recs;
{ initialize the cart, miscellaneous, and current values records           }

BEGIN
  fillchar( cart, sizeof( cart ), 0 );
  fillchar( misc, sizeof( misc ), 0 );
  fillchar( currvals, sizeof( currvals ), 0 );
  WITH cart DO
    BEGIN
      hoodvol := 421;                          lightinfo := 0.05;
      base_ptr := missing;                     air_on_line := false;
      on_line := false;                        new_air := true;
      carttype := hydroponic;                  ph_info.upperph := 14;
      co2info.optimum := 350;                  co2info.span := 500;
      FOR i := 1 TO maxchan DO
        conversions[ 2, i ] := 1.0;
    END;        { with cart... }
  WITH misc DO
    BEGIN
      airsamprate := 30;                       samprate := 300;
      transperiod := 30;                       pos := 1;
      chan := 1;                               numpoints := 6;
      errors.callok := false;
    END;
  WITH currvals DO
    BEGIN
      FOR i := 1 TO maxswitch DO
        switchvals[ i ] := false;
      airmeasure := false;                     regmeasure := false;
      answerok := false;                       stat_collect := false;
      modemonline := false;                    beep := false;
      data_collect := false;                   screennum := 1;
      statvolume := 12;                        workvolume := 11;
    END;
END;    { init_recs }

BEGIN   { init_parms }
  goxy( 0, 10 );
  prompt( 'Initializing new Control parameter file ' );
  { start up a new parameter file }
  str( vol, s );
  s := concat( '#', s, prm_extension );
  {$I-}
  rewrite( f, s );
  {$I+}
  iocode := ioresult;
  close( f, lock );
  IF iocode = 0 THEN
    BEGIN
      init_recs;
      FOR i := 0 TO curec DO
        BEGIN
          prompt( '.' );
          IF NOT( executed( update_parms( vol, i ) ) ) THEN
            i := curec + 1;
        END;
    END;        { if iocode... }
  init_parms := iocode;
  goxy( 0, 10 );
  clreoln;
END;    { init_parms }

BEGIN   { initialize }
  last_char := space;
  clrscrn;
  prompt( 'Define: ' );
  goxy( 0, 10 );
  prompt( 'Initializing' );
  ctrla := chr( 1 );
  ctrlb := chr( 2 );
  quit := 'Q';
  cmd_set := [ ctrla, ctrlb, quit, 'A'..'I' ];
  to_program := '';     vol := 0;
  pos_edit := posit;              eqn_edit := pos_eqns;
  get_calling_info( vol, to_program );
  IF vol = 0 THEN
    get_volume( vol, last_char );
  FOR i := 1 TO maxposition DO
    on_line[ i ] := false;
  IF last_char <> esc THEN
    IF executed( load_parms( vol, 0 ) ) THEN
      BEGIN
        goxy( 0, 10 );
        clreoln;
        prompt( 'Reading labels file' );
        perform( get_labels( vol, labels ) );
        goxy( 0, 10 );
        clreoln;
        prompt( 'Reading parameter file' );
        FOR i := 1 TO maxposition DO
          IF executed( load_parms( vol, i ) ) THEN
            BEGIN
              prompt( '.' );
              online[ i ] := cart.online;
            END;
      END
    ELSE
      IF NOT( executed( init_parms( vol ) ) ) THEN
        last_char := esc;
  goxy( 0, 10 );
  clreoln;
END;    { initialize }


PROCEDURE do_exit( to_program: string );

BEGIN
  clrscrn;
  goxy( 0, 10 );
  IF length( to_program ) > 0 THEN
    BEGIN
      prompt( concat( 'Returning to ', to_program, ' ' ) );
      setchain( to_program );
    END         { if length... }
  ELSE
    prompt( 'Returning to Pascal operating system ' );
  exit( PROGRAM );
END;    { do_exit }


PROCEDURE main_screen;
```

```
{ Display the program options                                        }

BEGIN
  promptln( 'Define Control parameters:' );
  goxy( 0, 5 );
  promptln( 'Edit:' );
  promptln( '-----' );
  promptln( 'A) Position parameters  [                ]' );
  promptln( 'B) Conversion equations [                              ]' );
  promptln( 'C) Current program values' );
  promptln( 'D) Miscellaneous program values' );
  promptln( 'E) Error parameters' );
  promptln( 'F) Channel parameters' );
  promptln( 'G) Transpiration rate parameters' );
  promptln( 'H) Last set of CO2 measurements (incl. std. measurements)' );
  promptln( '' );
  promptln( 'I) Help' );
  promptln( '' );
  promptln( '' );
  prompt( 'Q)uit, return to ' );
  IF length( to_program ) > 0 THEN
    prompt( to_program )
  ELSE
    prompt( 'Pascal operating system' );
  goxy( 25, 7 );
  sho_pos_edit_type( pos_edit );
  goxy( 25, 8 );
  sho_eqn_edit_type( eqn_edit );
  goxy( 0, 20 );
  prompt( 'Enter choice: ' );
END;    { main_screen }


PROCEDURE help;

BEGIN
END;    { help }


BEGIN   { main }
  initialize( last_char );
  IF last_char = esc THEN
    do_exit( to_program );
  clrscrn;
  main_screen;
  REPEAT
    goxy( 14, 20 );
    last_char := getchar( [ esc ] + cmd_set, false, false );
    CASE ord( last_char ) OF
      {A}       65: pos_parm( pos_edit );
      {B}       66: eqn_parm( eqn_edit );
      {C}       67: curr_parm;
      {D}       68: misc_parm;
      {E}       69: error_parm;
      {F}       70: chan_parm;
      {G}       71: trans_parm;
      {H}       72: std_parm;
      {I}       73: help;
      {ctrla}    1: BEGIN
                      IF pos_edit = posit THEN
                        pos_edit := param
                      ELSE
                        pos_edit := posit;
                      goxy( 25, 7 );
                      sho_pos_edit_type( pos_edit );
                    END;        { ctrla }
      {ctrlb}    2: BEGIN
                      IF eqn_edit = eqn THEN
```

```
                        eqn_edit := pos_eqns
                      ELSE
                        eqn_edit := succ( eqn_edit );
                      goxy( 25, 8 );
                      sho_eqn_edit_type( eqn_edit );
                    END;           { ctrlb }
    END;        { case }
    IF NOT( last_char IN [ ctrla, ctrlb, quit ] ) THEN
      BEGIN
        clrscrn;
        main_screen;
      END;
  UNTIL last_char IN [ quit, esc ];
  do_exit( to_program );
END.
```

```
UNIT define_stuff;
{-------------------------------------}
{ Define_stuff - a unit to be used with }
{ the Define program.  This unit        }
{ contains lower level editing          }
{ functions.                            }
{                                       }
{  4 Oct 85     0900           -mkm-    }
{-------------------------------------}

INTERFACE

USES
  {$U ctrlstuff.code} control_stuff;


TYPE
  pos_edit_type                               { edit by position or param }
                      =       ( posit, param );
  eqn_edit_type                               { edit by position, field,  }
                                              { or one equation only.     }
                      =       ( pos_eqns, field_eqns, eqn );

PROCEDURE get_bool( x, y: integer;            { screen location           }
                    VAR bool: boolean;        { answer                    }
                    VAR last_char: char );    { last character entered    }

FUNCTION get_int( x, y,                       { screen location           }
                  default,                    { default value             }
                  maxval,                     { maximum value allowed     }
                  minval: integer;            { minimum value allowed     }
                  VAR last_char: char )       { last character entered    }
                  : integer;

FUNCTION get_real( x, y: integer;             { screen location           }
                   default,                   { default value             }
                   maxval,                    { max value                 }
                   minval: real;              { min value                 }
                   VAR last_char: char )      { last char entered         }
                   : real;

PROCEDURE get_date( x, y: integer;            { screen location           }
                    default: ucsd_date;       { default date              }
                    VAR the_date: ucsd_date;  { answer                    }
                    VAR last_char: char );    { last char entered         }

PROCEDURE getstring( x, y: integer;           { screen location           }
                     default: string;         { default string            }
                     VAR s: string;           { answer                    }
                     okset: setofchar;        { allowable characters      }
                     maxlen: integer;         { max length of string      }
                     VAR last_char: char );   { last character entered    }

PROCEDURE get_kindcart( x, y: integer;        { screen location           }
                        default: kindcart;    { default                   }
                        VAR k: kindcart;      { answer                    }
                        VAR last_char: char ); { last character entered   }

PROCEDURE get_voltrange( x, y: integer;       { screen location           }
                         default: voltrange;  { default                   }
                         VAR v: voltrange;    { answer                    }
                         VAR last_char: char ); { last character entered  }

PROCEDURE get_period_type( x, y: integer;     { screen location           }
                           default: period_type;{ default value           }
                           VAR p: period_type;  { answer                  }
                           VAR last_char: char );
```

```
PROCEDURE get_airtype( x, y: integer;          { screen location           }
                       default: airtype;       { default value             }
                       VAR a: airtype;         { answer                    }
                       VAR last_char: char );

PROCEDURE sho_kindcart( k: kindcart );

PROCEDURE sho_voltrange( v: voltrange );

PROCEDURE sho_period_type( p: period_type );

PROCEDURE sho_airtype( a: airtype );

PROCEDURE sho_pos_edit_type( p: pos_edit_type );

PROCEDURE sho_eqn_edit_type( e: eqn_edit_type );

PROCEDURE sho_bool( b: boolean );

PROCEDURE sho_date( d: ucsd_date );

PROCEDURE update_subscript( last_char: char; VAR subscript: integer;
                                         max, x_inc, y_inc: integer );


IMPLEMENTATION


VAR
  cart_names    :      ARRAY[ kindcart ] OF string[ 10 ];
  vr_names      :      ARRAY[ voltrange ] OF string[ 8 ];
  period_names  :      ARRAY[ period_type ] OF string[ 16 ];
  air_names     :      ARRAY[ airtype ] OF string[ 7 ];
  pos_names     :      ARRAY[ pos_edit_type ] OF string[ 12 ];
  eqn_names     :      ARRAY[ eqn_edit_type ] OF string[ 26 ];
  date_set,                                    { characters for date entry }
  tog_set       :      set_of_char;            { legal chars for toggles   }

PROCEDURE ret_prompt;
{ prompt for boolean and special type selection }

BEGIN
  goxy( 0, error_line );
  clreoln;
  prompt( 'Press <ret> to toggle' );
END;    { ret_prompt }


FUNCTION getfpnum( default, maxval, minval: real; VAR lastchar:char ):real;
{ gets floating point number between specified max and min values }

BEGIN
  getfpnum := getnum( default, maxval, minval, true, last_char );
END;  { getfpnum }


PROCEDURE get_bool( x, y: integer, VAR bool: boolean; VAR last_char: char );

BEGIN
  ret_prompt;
  goxy( x, y );
  last_char := getchar( tog_set + [ 'Y', 'N', 'T', 'F' ], false, false );
  IF last_char = ret THEN
    bool := NOT bool
  ELSE IF last_char IN [ 'Y', 'T' ] THEN
    bool := true
  ELSE IF last_char IN [ 'N', 'F' ] THEN
```

```
    bool := false;
END;


FUNCTION get_int( ( x, y, default, maxval, minval: integer;
                    VAR last_char: char ): integer );

BEGIN
  goxy( 0, error_line );
  clreoln;
  prompt( 'Enter a value between ' );
  prompt_real( minval, 0, 0 );
  prompt( ' and ' );
  prompt_real( maxval, 0, 0 );
  goxy( x, y );
  get_int := get_integer( default, maxval, minval, last_char );
END;    { get_int }


FUNCTION get_real((x, y: integer; default, maxval, minval: real;
                   VAR last_char: char ): real);

BEGIN
  goxy( 0, error_line );
  clreoln;
  prompt( 'Enter a value between ' );
  prompt_real( minval, 8, 2 );
  prompt( ' and ' );
  prompt_real( maxval, 8, 2 );
  goxy( x, y );
  get_real := get_fpnum( default, maxval, minval, last_char );
END;    { get_real }


PROCEDURE get_date( x, y: integer; default: ucsd_date;
                    VAR the_date: ucsd_date; VAR last_char: char );

VAR
  def_date,
  new_date      :       string;

PROCEDURE upstring( VAR s: string );
{ put s into upper case }

VAR
  i : integer;
  ch: char;

BEGIN
  IF length( s ) > 0 THEN
    FOR i := 1 TO length( s ) DO
      BEGIN
        ch := s[ i ];  upcase( ch );  s[ i ] := ch;
      END;
END;    { upstring }

PROCEDURE string_2_date( date_string: string; VAR date: ucsd_date );
{ Convert supposed date in date string to ucsd format.  }
{ Date does not change on error                         }

CONST
  delim         =       '-';

VAR
  s,
  t,
  mo_str:       string;
  err   :       boolean;
  temp,
  position:     integer;

BEGIN
  IF length( date_string ) > 0 THEN
    BEGIN
      mo_str := month_names;
      delete( mo_str, 1, 1 );
      upstring( mo_str );
      err := false;
      position := pos( delim, date_string );
      IF position = 0 THEN
        t := date_string
      ELSE
        t := copy( date_string, 1, position - 1 );  { should be the day }
      IF length( t ) > 0 THEN
        temp := trunc( fpnum( t ) )
      ELSE temp := 0;
      IF temp IN [ 1..31 ] THEN
        date.day := temp
      ELSE
        err := true;
      IF NOT err THEN           { get the month }
        BEGIN
          delete( date_string, 1, position );
          position := pos( delim, date_string );
          err := length( date_string ) = 0;
          IF NOT err THEN
            BEGIN
              IF position = 0 THEN
                t := date_string
              ELSE
                t := copy( date_string, 1, position - 1 );
              upstring( t );
              temp := pos( t, mo_str ) DIV 3;
              IF temp IN [ 1..12 ] THEN
                date.mo := temp
              ELSE
                err := true;
            END;         { if not err }
        END;    { if not err }
      IF NOT err THEN           { set the year }
        BEGIN
          delete( date_string, 1, position );
          IF length( date_string ) > 0 THEN
            BEGIN
              temp := trunc( fpnum( date_string ) );
              IF temp IN [ 0..100 ] THEN
                date.yr := temp;
            END;         { if length( date_string )... }
        END;    { if not err }
    END;        { if length( date_string )... }
END;    { string_2_date }

BEGIN   { get_date }
  the_date := default;
  date_2_string( default, def_date );
  get_string( x, y, def_date, new_date, date_set, 9, last_char );
  string_2_date( new_date, the_date );
END;    { get_date }


PROCEDURE getstring( x, y: integer; default: string; var s: string;
                     okset: setofchar; maxlen: integer; last_char: char );
{ gets string with maximum length and characters in okset }

VAR s1  :       char;
    stemp :     string;
    len :       integer;
    done,
    firstchar,
    endchar  : boolean;
    getset    : setofchar;

 BEGIN
   goxy( 0, error_line );
   clreoln;
   prompt( 'Enter a string with a max length of ' );
   prompt_real( maxlen, 0, 0 );
   prompt( ' characters' );
   goxy( x, y );
   s1 := space;  stemp := '';  last_char := space;
   IF maxlen < 1 THEN maxlen := 1
   ELSE IF maxlen > 255 THEN maxlen := 255;
   done := false;
   REPEAT
     len := length( stemp );
     firstchar := ( len = 0 );
     endchar := ( len = maxlen );

     IF firstchar THEN getset := okset + exit_set + cmd_set + [ keyleft ]
     ELSE IF NOT endchar THEN getset := okset + exit_set + [ keyleft ]
     ELSE getset := exit_set + [ keyleft ];

     s1 := getchar( getset, true, true );

     IF s1 = esc THEN stemp := '';

     done := finished( firstchar, s1, [] );

     IF done THEN
       BEGIN
         up_case( s1 );
         last_char := s1;
       END;      { if done }

     IF ( s1 IN okset ) AND ( NOT done ) THEN
       BEGIN
         stemp := concat( stemp, ' ' );
         stemp[ length( stemp ) ] := s1;
       END
     ELSE IF ( s1 = keyleft ) AND ( NOT done ) THEN
       BEGIN
         write( keyleft, space, keyleft);
         delete( stemp, len, 1);
       END;

   UNTIL done;  promptln( '' );
   IF length( stemp ) > 0 THEN
     s := stemp
   ELSE s := default;
 END;  { getstring }


 PROCEDURE get_kindcart( x, y: integer; default: kindcart; VAR k: kindcart;
                                                   VAR last_char: char );
 BEGIN
   ret_prompt;
   goxy( x, y );
   last_char := getchar( tog_set, false, false );
   IF last_char = ret THEN
     IF k = extra2 THEN
       k := hydroponic
     ELSE
       k := succ( k )
   ELSE
```

```
    k := default;
END;    { get_kindcart }


PROCEDURE get_voltrange( x, y: integer; default: voltrange; VAR v: voltrange;
                                                  VAR last_char: char );

BEGIN
  ret_prompt;
  goxy( x, y );
  last_char := getchar( tog_set, false, false );
  IF last_char = ret THEN
    IF v = disconnect THEN
      v := plus5v
    ELSE
      v := succ( v )
  ELSE
    v := default;
END;    { get_voltrange }


PROCEDURE get_period_type( x, y: integer; default: period_type;
                                 VAR p: period_type; VAR last_char: char );
BEGIN
  ret_prompt;
  goxy( x, y );
  last_char := getchar( tog_set, false, false );
  IF last_char = ret THEN
    IF p = day THEN
      p := tran_to_nite
    ELSE
      p := succ( p )
  ELSE
    p := default;
END;    { get_period_type }


PROCEDURE get_airtype( x, y: integer; default: airtype; VAR a: airtype;
                                                  VAR last_char: char );
BEGIN
  ret_prompt;
  goxy( x, y );
  last_char := getchar( tog_set, false, false );
  IF last_char = ret THEN
    IF a = gasmeas THEN
      a := fill
    ELSE
      a := succ( a )
  ELSE
    a := default;
END;    { get_airtype }


PROCEDURE sho_kindcart( k: kindcart );

BEGIN
  prompt( cart_names[ k ] );
END;    { sho_kindcart }


PROCEDURE sho_voltrange( v: voltrange );

BEGIN
  prompt( vr_names[ v ] );
END;    { sho_voltrange }


PROCEDURE sho_period_type( p: period_type );
```

```
BEGIN
  prompt( period_names[ p ] );
END;    { sho_period_type }


PROCEDURE sho_airtype( a: airtype );

BEGIN
  prompt( air_names[ a ] );
END;    { sho_airtype }


PROCEDURE sho_pos_edit_type( p: pos_edit_type );

BEGIN
  prompt( pos_names[ p ] );
END;    { sho_pos_edit_type }


PROCEDURE sho_eqn_edit_type( e: eqn_edit_type );

BEGIN
  prompt( eqn_names[ e ] );
END;    { sho_eqn_edit_type }


PROCEDURE sho_bool( b: boolean );

BEGIN
  IF b THEN
    BEGIN
      evon;
      prompt( 'Y' );
      evoff;
    END
  ELSE
    prompt( 'N' );
END;    { sho_bool }


PROCEDURE sho_date( d: ucsd_date );

VAR
  date_str      :       string;

BEGIN
  date_2_str( d, date_str );
  prompt( date_str );
END;    { sho_date }


PROCEDURE update_subscript( last_char: char; VAR subscript: integer;
                                              max, x_inc, y_inc: integer );
{ change the subscript based on the last character pressed }

BEGIN
  IF last_char = keyright THEN
    subscript := subscript + x_inc
  ELSE IF last_char = keyleft THEN
    subscript := subscript - x_inc
  ELSE IF last_char = keyup THEN
    subscript := subscript - y_inc
  ELSE IF last_char = keydown THEN
    subscript := subscript + y_inc;

  IF subscript > max THEN
    subscript := subscript - max
    ELSE IF subscript < 1 THEN
      subscript := subscript + max ;

    IF subscript > max THEN
      subscript := 1
    ELSE IF subscript < 1 THEN
      subscript := max;
  END;    { update_subscript }


  BEGIN   { unit initialization }
    vr_names[ plus5v ]:= '+5.0 v  ';        vr_names[ plus1v ]:= '+1.0 v  ';
    vr_names[ plus500mv ]:= '+500mv  ';     vr_names[ plus100mv ]:= '+100mv  ';
    vr_names[ dual5v ]:= '+/-5.0v ';       vr_names[ dual1v ]:= '+/-1.0v ';
    vr_names[ dual500mv ]:= '+/-500mv';   vr_names[ dual100mv ]:= '+/-100mv';
    vr_names[ disconnect ]:= 'disconn.';
    cart_names[ hydroponic ] := 'hydroponic';
    cart_names[ soil ]       := '      soil';
    cart_names[ extra1 ]     := '     extra';
    cart_names[ extra2 ]     := '     extra';
    period_names[ tran_to_nite ] := 'transition/night';
    period_names[ night ]        := '           night';
    period_names[ tran_to_day ]  := ' transition/day';
    period_names[ day ]          := '             day';
    air_names[ fill ]    := '   fill';
    air_names[ purge ]   := '  purge';
    air_names[ gasmeas ] := 'measure';
    pos_names[ posit ] := ' by position';
    pos_names[ param ] := 'by parameter';
    eqn_names[ pos_eqns ]   := 'one position/all equations';
    eqn_names[ field_eqns ] := 'one equation/all positions';
    eqn_names[ eqn ]        := '              one equation';
    tog_set := [ keyleft, keyright, keyup, keydown, ret, esc ];
    date_set := [ 'A'..'Y', 'a'..'y', '0'..'9', '-' ];
  END.
```

```
{---------------------------------------------}
{ Define1.text  - an include file for the     }
{ Define program.                             }
{                                             }
{ segment procedures - pos, eqn, curr.        }
{                                             }
{ 11 Oct 85     1600                   -mkm-  }
{---------------------------------------------}

segment PROCEDURE pos_parm( pos_edit: pos_edit_type );
{ edit the position records, cart }

VAR
  temp_set      :       setofchar;      { temp cmd set, restore when done   }
  last_char     :       char;

PROCEDURE edit_by_position;
{ edit all the parameters for one position }

CONST
  pos_screen    =       5;              { display screen #5                 }
  max_subscript =       30;             { maximum subscript for editing     }

VAR
  position      :       integer;

PROCEDURE get_position( VAR position: integer );
{ get the position record to edit }

VAR
  test          :       integer;

FUNCTION next( position: integer ): integer;
{ calculate the next highest on-line position }

BEGIN
  position := position + 1;
  IF position > maxposition THEN
    position := 1;
  next := position;
END;    { next }

FUNCTION before( position: integer ): integer;
{ calculate the next lowest on_line position }

BEGIN
  position := position - 1;
  IF position < 1 THEN
    position := maxposition;
  before := position;
END;    { before }

BEGIN   { get_position }
  clrscrn;
  goxy( 0, 5 );
  promptln( 'Choose position to edit: [  ]' );
  promptln( '' );
  promptln( '' );
  promptln( '<ret> to accept position; Enter new position or use <--|-->' );
  promptln( 'for next on-line position.  <esc> to quit' );
  REPEAT
    goxy( 26, 5 );
    write( position:2 );
    goxy( 26, 5 );
    position := getinteger( position, maxposition, 1, last_char );
    IF last_char = keyright THEN
      BEGIN
```

```
        test := position;
        position := next( position );
        WHILE ( position <> test ) AND ( NOT( online[ position ] ) ) DO
          position := next( position );
      END       { if last_char... }
    ELSE IF last_char = keyleft THEN
      BEGIN
        test := position;
        position := before( position );
        WHILE ( position <> test ) AND ( NOT( online[ position ] ) ) DO
          position := before( position );
      END;      { else if last_char... }
  UNTIL ( last_char IN [ ret, esc ] ) AND ( position IN [ 1..maxposition ] );
END;    { get_position }

PROCEDURE find_x_y( subscript: integer; VAR x, y: integer );

BEGIN
  IF subscript IN [ 1..15 ] THEN
    BEGIN
      x := 26;
      y := subscript + 2;
    END
  ELSE
    BEGIN
      x := 66;
      y := subscript - 13;
    END;
END;    { find_x_y }

PROCEDURE move( subscript: integer );

VAR
  x,
  y     :       integer;

BEGIN
  find_x_y( subscript, x, y );
  goxy( x, y );
END;    { move }

PROCEDURE show( subscript: integer );

BEGIN
  WITH cart DO
    CASE subscript OF
      1:  prompt_real( cartnum, 2, 0 );
      2:  prompt_real( num_plants, 2, 0 );
      3:  prompt_real( lite_pos, 2, 0 );
      4:  prompt_real( added_today, 6, 0 );
      5:  prompt_real( base_ptr, 6, 0 );
      6:  sho_bool( aironline );
      7:  sho_bool( online );
      8:  sho_bool( newair );
      9:  prompt_real( hoodvol, 8, 2 );
      10: prompt_real( lightinfo, 8, 2 );
      11: prompt_real( currrate, 8, 2 );
      12: prompt_real( lastco2, 8, 2 );
      13: prompt_real( tottranspired, 8, 2 );
      14: prompt_real( last_lite, 8, 2 );
      15: sho_kindcart( carttype );
      16: prompt_real( phinfo.upperph, 8, 2 );
      17: prompt_real( phinfo.lowerph, 8, 2 );
      18: prompt_real( phinfo.timetoadd, 2, 0 );
      19: prompt_real( solninfo.transpired, 8, 2 );
      20: prompt_real( solninfo.solution, 8, 2 );
      21: prompt_real( solninfo.amount, 4, 0 );
      22: prompt_real( co2info.optimum, 8, 2 );
      23: prompt_real( co2info.span, 8, 2 );
      24: sho_bool( dosed );
      25: prompt_real( dose_time.lasttime, 4, 0 );
      26: prompt_real( dose_time.lastsecs, 2, 0 );
      27: sho_date( dose_time.lastdate );
      28: prompt_real( rate_time.lasttime, 4, 0 );
      29: prompt_real( rate_time.lastsecs, 2, 0 );
      30: sho_date( rate_time.lastdate );
    END;  { case }
END;  { show }

PROCEDURE sho_parms( position: integer );

VAR
  subscript: integer;

BEGIN
  clrscrn;
  dispscreen( pos_screen );
  goxy( 0, error_line + 2 );
  evon;
  prompt( 'Press <esc> to quit' );
  evoff;
  goxy( 41, 1 );
  prompt_real( position, 2, 0 );
  FOR subscript := 1 TO max_subscript DO
    BEGIN
      move( subscript );
      show( subscript );
    END;
END;    { sho_parms }

PROCEDURE edit_parms;
{ edit one cart variable }

VAR
  x,
  y,
  subscript     :       integer;

PROCEDURE get_online( x, y: integer; VAR last_char: char );

BEGIN
  get_bool( x, y, cart.online, last_char );
  IF cart.online THEN online[ position ] := true
  ELSE online[ position ] := false;
END;    { get_online }

BEGIN   { edit_parms }
  subscript := 1;
  REPEAT
    move( subscript );
    find_x_y( subscript, x, y );
    WITH cart DO
      CASE subscript OF
        1:  cartnum := get_int( x, y, cartnum, maxcart, 1, last_char );
        2:  num_plants := get_int( x, y, num_plants, 99, 0, last_char );
        3:  lite_pos := get_int( x, y, lite_pos, max_position, 1, last_char );
        4:  added_today := get_int( x, y, added_today, maxint, 0, last_char );
        5:  base_ptr := get_int( x, y, base_ptr, maxstatrec,
                                                      missing, last_char );
        6:  get_bool( x, y, cart.aironline, last_char );
        7:  get_online( x, y, last_char );
        8:  get_bool( x, y, cart.newair, last_char );
        9:  hood_vol := get_real( x, y, hood_vol, max_real, 0, last_char );
        10: lightinfo := get_real( x, y, light_info, max_real, min_real,
                                                             last_char );
        11: currrate := get_real( x, y, currrate, max_real, min_real,
                                                                  last_char );
          12: last_co2 := get_real( x, y, last_co2, max_real, min_real,
                                                                  last_char );
          13: tottranspired := get_real( x, y, tottranspired, max_real, min_real,
                                                                  last_char );
          14: last_lite := get_real( x, y, last_lite, max_real, min_real,
                                                                  last_char );
          15: get_kindcart( x, y, carttype, carttype, last_char );
          16: WITH phinfo DO
                upperph := get_real( x, y, upperph, 14, 1, lastchar );
          17: WITH phinfo DO
                lowerph := get_real( x, y, lowerph, 14, 1, lastchar );
          18: WITH phinfo DO
                timetoadd := getint( x, y, timetoadd, maxint, 0, lastchar );
          19: WITH solninfo DO
                transpired := getreal( x, y, transpired, maxreal, 0, last_char );
          20: WITH solninfo DO
                solution := getreal( x, y, solution, maxreal, 0, last_char );
          21: WITH solninfo DO
                amount := getint( x, y, amount, maxint, 0, last_char );
          22: WITH co2info DO
                optimum := getreal( x, y, optimum, 10000, 0, last_char );
          23: WITH co2info DO
                span := getreal( x, y, span, 10000, 0, lastchar );
          24: get_bool( x, y, dosed, last_char );
          25: WITH dosetime DO
                lasttime := getint( x, y, lasttime, 5959, 0, last_char );
          26: WITH dosetime DO
                lastsecs := getint( x, y, lastsecs, 59, 0, lastchar );
          27: getdate( x, y, dosetime.lastdate, dosetime.lastdate, lastchar );
          28: WITH ratetime DO
                lasttime := getint( x, y, lasttime, 5959, 0, last_char );
          29: WITH ratetime DO
                lastsecs := getint( x, y, lastsecs, 59, 0, lastchar );
          30: getdate( x, y, ratetime.lastdate, ratetime.lastdate, lastchar );
        END;  { case }
      move( subscript );
      show( subscript );
      update_subscript( last_char, subscript, maxsubscript, 1, 1 );
    UNTIL ( last_char = esc );
  END;    { edit_parms }

  BEGIN   { edit_by_position }
    position := 1;
    REPEAT
      get_position( position );
      IF last_char <> esc THEN
        IF executed( load_parms( vol, position ) ) THEN   { loads cart var    }
          BEGIN
            sho_parms( position );
            edit_parms;
            perform( update_parms( vol, position ) );
            last_char := space;
          END;    { if last_char, if executed... }
    UNTIL last_char = esc;
  END;    { edit_by_position }

  PROCEDURE edit_by_parameter;
  { edit one parameter for all the positions }

  BEGIN   { edit_by_parameter }
    (* get_parm( last_char );
    IF last_char <> esc THEN *)
  END;    { edit_by_parameter }

  BEGIN   { segment pos_parm }
    temp_set := cmd_set;
    IF pos_edit = posit THEN
    edit_by_position
  ELSE
    edit_by_parameter;
  cmd_set := temp_set;
END;    { segment pos_parm }


segment PROCEDURE eqn_parm( eqn_edit: eqn_edit_type );

VAR
  temp_set      :       setofchar;
  last_char     :       char;

PROCEDURE get_position( VAR position: integer );
{ get the position record to edit }

VAR
  test          :       integer;

FUNCTION next( position: integer ): integer;
{ calculate the next highest on-line position }

BEGIN
  position := position + 1;
  IF position > maxposition THEN
    position := 1;
  next := position;
END;    { next }

FUNCTION before( position: integer ): integer;
{ calculate the next lowest on_line position }

BEGIN
  position := position - 1;
  IF position < 1 THEN
    position := maxposition;
  before := position;
END;    { before }

BEGIN   { get_position }
  clrscrn;
  goxy( 0, 5 );
  promptln( 'Choose position to edit: [  ]' );
  promptln( '' );
  promptln( '' );
  promptln( '<ret> to accept position; Enter new position or use <--|-->' );
  promptln( 'for next on-line position.  <esc> to quit' );
  REPEAT
    goxy( 26, 5 );
    write( position:2 );
    goxy( 26, 5 );
    position := getinteger( position, maxposition, 1, last_char );
    IF last_char = keyright THEN
      BEGIN
        test := position;
        position := next( position );
        WHILE ( position <> test ) AND ( NOT( online[ position ] ) ) DO
          position := next( position );
      END       { if last_char... }
    ELSE IF last_char = keyleft THEN
      BEGIN
        test := position;
        position := before( position );
        WHILE ( position <> test ) AND ( NOT( online[ position ] ) ) DO
          position := before( position );
      END;      { else if last_char... }
  UNTIL ( last_char IN [ ret, esc ] ) AND ( position IN [ 1..maxposition ] );
END;    { get_position }
```

```
PROCEDURE edit_pos;
{ allows editing of all the equations for one position }

CONST
  max_sub       =       80;
  scrn          =       14;

VAR
  position      :       integer;

PROCEDURE find_x_y( subscript: integer; VAR x, y: integer );

BEGIN
  y := ( ( subscript - 1 ) DIV 5 ) + 4;
  CASE ( subscript MOD 5 ) OF
    1: x := 11;
    2: x := 25;
    3: x := 39;
    4: x := 53;
    0: x := 67;
  END;  { case }
END;    { find_x_y }

PROCEDURE find_i_j( subscript: integer; VAR i, j: integer );
{ find the subscript to the conversion variable }

BEGIN
  i := ( ( subscript - 1 ) MOD 5 ) + 1;
  j := ( ( subscript - 1 ) DIV 5 ) + 1;
END;    { find_i_j }

PROCEDURE move( subscript: integer );

VAR
  x, y          :       integer;

BEGIN
  find_x_y( subscript, x, y );
  goxy( x, y );
END;    { move }

PROCEDURE show( subscript: integer );
{ display the appropriate conversion parameter }

VAR
  i,
  j     :       integer;

BEGIN
  find_i_j( subscript, i, j );
  WITH cart DO
    prompt_real( conversions[ i, j ], 8, 4 );
END;    { show }

PROCEDURE sho_parms( position: integer );
{ display all the conversion equations for this position }

VAR
  i     :       integer;

BEGIN
  clrscrn;
  dispscreen( scrn );
  goxy( 0, error_line + 2 );
  evon;
  prompt( 'Press <esc> to quit' );
  evoff;
```

```
    goxy( 29, 0 );
    prompt_real( position, 2, 0 );
    goxy( 43, 0 );
    prompt_real( cart.cartnum, 2, 0 );
    FOR i := 1 TO max_sub DO
      BEGIN
        move( i );
        show( i );
      END;
  END;    { sho_parms }

  PROCEDURE edit_parms;

  VAR
    subscript,
    x, y,
    i, j  :       integer;

  BEGIN
    subscript := 1;
    REPEAT
      find_i_j( subscript, i, j );
      find_x_y( subscript, x, y );
      move( subscript );
      WITH cart DO
        conversions[ i, j ] := get_real( x, y, conversions[ i, j ],
                                         maxreal, minreal, last_char );
      move( subscript );
      show( subscript );
      update_subscript( last_char, subscript, max_sub, 1, 1 );
    UNTIL last_char = esc;
  END;    { edit_parms }

  BEGIN   { edit_pos }
    position := 1;
    REPEAT
      get_position( position );
      IF last_char <> esc THEN
        IF executed( load_parms( vol, position ) ) THEN   { loads cart var    }
          BEGIN
            sho_parms( position );
            edit_parms;
            perform( update_parms( vol, position ) );
            last_char := space;
          END;    { if last_char, if executed... }
    UNTIL last_char = esc;
  END;    { edit_pos }

  PROCEDURE edit_eqns;
  { allows editing of one equation for all the positions }

  BEGIN
  END;    { edit_eqns }

  PROCEDURE edit_one;
  { allows editing of one equation for one position }

  BEGIN
  END;    { edit_one }

  BEGIN   { segment eqn_parm }
    IF eqn_edit = pos_eqns THEN
      edit_pos
    ELSE IF eqn_edit = field_eqns THEN
      edit_eqns
    ELSE
      edit_one;
  END;    { segment eqn_parm }
```

```
segment PROCEDURE curr_parm;
{ edit the current program values record }

CONST
  max_sub       =       32;
  cu_screen     =       13;

VAR
  last_char     :       char;
  temp_set      :       setofchar;

PROCEDURE find_x_y( subscript: integer; VAR x, y: integer );

BEGIN
  IF subscript IN [ 1..16 ] THEN
    BEGIN
      x := ( subscript - 1 ) * 4 + 1;
      y := 4;
    END { if subscript in... }
  ELSE IF subscript IN [ 17..27 ] THEN
    BEGIN
      x := 29;
      y := subscript - 11;
    END
  ELSE
    BEGIN
      x := 70;
      y := subscript - 22;
    END;
END;    { find_x_y }

PROCEDURE move( subscript: integer );

VAR
  x, y  :       integer;

BEGIN
  find_x_y( subscript, x, y );
  goxy( x, y );
END;    { move }

PROCEDURE show( subscript: integer );

BEGIN
  WITH currvals DO
    BEGIN
      IF subscript IN [ 1..16 ] THEN
        sho_bool( switchvals[ subscript ] )
      ELSE CASE subscript OF
        17: sho_airtype( aircondition );
        18: prompt_real( gascart, 2, 0 );
        19: prompt_real( skipcount, 4, 0 );
        20: sho_bool( airmeasure );
        21: sho_bool( regmeasure );
        22: sho_bool( answerok );
        23: sho_bool( stat_collect );
        24: sho_bool( modem_on_line );
        25: prompt( airstring );
        26: sho_period_type( currperiod );
        27: sho_period_type( lastperiod );
        28: sho_bool( beep );
        29: sho_bool( data_collect );
        30: prompt_real( screennum, 1, 0 );
        31: prompt_real( statvolume, 2, 0 );
        32: prompt_real( workvolume, 2, 0 );
      END;      { case }
    END;        { with currvals do... }
END;    { show }

PROCEDURE sho_parms;

VAR
  subscript     :       integer;

BEGIN
  clrscrn;
  dispscreen( cu_screen );
  goxy( 0, error_line + 2 );
  evon;
  prompt( 'Press <esc> to quit' );
  evoff;
  FOR subscript := 1 TO max_sub DO
    BEGIN
      move( subscript );
      show( subscript );
    END;        { for }
END;    { sho_parms }

PROCEDURE edit_parms;

VAR
  x, y,
  subscript     :       integer;

PROCEDURE get_new_volume( VAR vol: integer; x, y: integer;
                                                        VAR last_char: char );
{ gets a legal volume number in 4, 5, 9..12.
                                                                            }

VAR
  test  :       integer;

BEGIN
  test := vol;
  vol := get_int( x, y, vol, 12, 4, last_char );
  IF NOT( vol IN [ 4, 5, 9..12 ] ) THEN
    BEGIN
      beep;
      vol := test;
      last_char := ret;
    END;
END;    { get_new_volume }

BEGIN   { edit_parms }
  subscript := 1;
  REPEAT
    find_x_y( subscript, x, y );
    move( subscript );
    IF subscript IN [ 1..16 ] THEN
      get_bool( x, y, currvals.switchvals[ subscript ], last_char )
    ELSE WITH currvals DO
      CASE subscript OF
        17: get_airtype( x, y, aircondition, aircondition, last_char );
        18: gascart := get_int( x, y, gascart, maxposition, -1, last_char );
        19: skipcount := get_int( x, y, skipcount, 9999, 0, last_char );
        20: get_bool( x, y, airmeasure, last_char );
        21: get_bool( x, y, regmeasure, last_char );
        22: get_bool( x, y, answerok, last_char );
        23: get_bool( x, y, stat_collect, last_char );
        24: get_bool( x, y, modem_on_line, last_char );
        25: {$v-}
            get_string( x, y, airstring, airstring, [ ' '..'~' ], 30,
                                                                last_char );
            {$v+}
        26: get_period_type( x, y, currperiod, currperiod, last_char );
        27: get_period_type( x, y, lastperiod, lastperiod, last_char );
        28: get_bool( x, y, beep, last_char );
        29: get_bool( x, y, data_collect, last_char );
        30: screennum := get_int( x, y, screennum, 3, 1, last_char );
        31: get_new_volume( statvolume, x, y, last_char );
        32: get_new_volume( workvolume, x, y, last_char );
      END;      { with, case.... }
    move( subscript );
    show( subscript );
    update_subscript( last_char, subscript, max_sub, 1, 1 );
    IF ( subscript IN [ 1..16 ] ) AND ( last_char = keydown ) THEN
      subscript := 17;
  UNTIL last_char = esc;
END;    { edit_parms }

BEGIN   { segment curr_parm }
  IF executed( load_parms( vol, curec ) ) THEN
    BEGIN
      sho_parms;
      edit_parms;
    END;
  perform( update_parms( vol, curec ) );
END;    { segment curr_parm }
```

```
{-----------------------------------------------}
{ Define2.text  - an include file for the       }
{ Define program.                               }
{                                               }
{ more segment procedures - misc, error, chan   }
{ trans, std.                                   }
{                                               }
{ 11 Oct 85     1600                   -mkm-    }
{-----------------------------------------------}


segment PROCEDURE misc_parm;
{ edit the miscellaneous program values record }

CONST
  max_sub       =       45;
  mi_screen     =       9;

VAR
  last_char     :       char;
  temp_set      :       setofchar;

PROCEDURE find_x_y( subscript: integer; VAR x, y: integer );

BEGIN
  IF subscript IN [ 1..18 ] THEN
    BEGIN
      y := 4;
      x := ( subscript - 1 ) * 4 + 1;
    END
  ELSE IF subscript IN [ 19..maxfield ] THEN
    BEGIN
      y := 6;
      x := ( subscript - 19 ) * 4 + 1;
    END
  ELSE
    BEGIN
      x := 57;
      y := subscript - 29;
    END;        { if then elses... }
END;    { find_x_y }

PROCEDURE move( subscript: integer );

VAR
  x, y  :       integer;

BEGIN
  find_x_y( subscript, x, y );
  goxy( x, y );
END;    { move }

PROCEDURE show( subscript: integer );

BEGIN
  WITH misc DO
    BEGIN
      IF subscript IN [ 1..maxfield ] THEN
        sho_bool( collect_stats[ subscript ] )
      ELSE CASE subscript OF
      37: prompt_real( flowrate, 8, 2 );
      38: prompt_real( airsamprate, 4, 0 );
      39: prompt_real( samprate, 4, 0 );
      40: prompt_real( transperiod, 4, 0 );
      41: prompt_real( pos, 2, 0 );
      42: prompt_real( chan, 2, 0 );
      43: prompt_real( skip, 4, 0 );
      44: prompt_real( numpoints, 4, 0 );
      END;         { else, case... }
    END;  { with misc... }
  END;    { show }

  PROCEDURE sho_parms;

  VAR
    subscript     :       integer;

  BEGIN
    clrscrn;
    dispscreen( mi_screen );
    goxy( 0, error_line + 2 );
    evon;
    prompt( 'Press <esc> to quit' );
    evoff;
    FOR subscript := 1 TO max_sub DO
      BEGIN
        move( subscript );
        show( subscript );
      END;         { for }
  END;    { sho_parms }

  PROCEDURE edit_parms;

  VAR
    x, y,
    subscript     :       integer;

  BEGIN
    subscript := 1;
    REPEAT
      WITH misc DO
        BEGIN
          find_x_y( subscript, x, y );
          move( subscript );
          IF subscript IN [ 1..maxfield ] THEN
            get_bool( x, y, collect_stats[ subscript ], last_char )
          ELSE CASE subscript OF
            37: flowrate := get_real( x, y, flowrate, maxreal, 0, last_char );
            38: airsamprate := get_int( x, y, airsamprate, 9999, 0, last_char );
            39: samprate := get_int( x, y, samprate, 9999, 15, last_char );
            40: transperiod := get_int( x, y, transperiod, 9999, 0, last_char );
            41: pos := get_int( x, y, pos, maxposition, 1, last_char );
            42: chan := get_int( x, y, chan, maxfield, 1, last_char );
            43: skip := get_int( x, y, skip, 9999, 0, last_char );
            44: numpoints := get_int( x, y, numpoints, 6, 2, last_char );
          END;         { case }
          move( subscript );
          show( subscript );
          IF ( subscript IN [ 19..maxfield ] ) AND ( last_char = keydown ) THEN
            subscript := 37
          ELSE IF ( subscript IN [ 1..18 ] ) AND ( last_char = keyup ) THEN
            subscript := maxsub
          ELSE IF subscript IN [ 1..maxfield ] THEN
            update_subscript( last_char, subscript, max_sub, 1, 18 )
          ELSE
            update_subscript( last_char, subscript, max_sub, 1, 1 );
        END;      { with... }
    UNTIL last_char = esc;
  END;    { edit_parms }

BEGIN   { segment misc_parm }
  IF executed( load_parms( vol, mirec ) ) THEN
    BEGIN
      sho_parms;
      edit_parms;
    END;
  perform( update_parms( vol, mirec ) );
END;    { segment misc_parm }


segment PROCEDURE error_parm;
{ edit the errors section of the miscellaneous program values record }

CONST
  max_sub       =       38;
  error_screen     =    8;

VAR
  last_char     :       char;
  temp_set      :       setofchar;

PROCEDURE find_x_y( subscript: integer; VAR x, y: integer );

BEGIN
  IF subscript IN [ 1..8 ] THEN
    BEGIN
      x := 16;
      y := subscript + 3;
    END
  ELSE IF subscript IN [ 9..16 ] THEN
    BEGIN
      x := 26;
      y := subscript - 5;
    END
  ELSE IF subscript IN [ 17..24 ] THEN
    BEGIN
      x := 56;
      y := subscript - 13;
    END
  ELSE IF subscript IN [ 25..32 ] THEN
    BEGIN
      x := 66;
      y := subscript - 21;
    END
  ELSE
    BEGIN
      x := 49;
      y := subscript - 20;
    END;        { if then elses... }
END;    { find_x_y }

PROCEDURE move( subscript: integer );

VAR
  x, y  :       integer;

BEGIN
  find_x_y( subscript, x, y );
  goxy( x, y );
END;    { move }

PROCEDURE show( subscript: integer );

BEGIN
  WITH misc.errors DO
    BEGIN
      IF subscript IN [ 1..8 ] THEN
        prompt_real( errorranges[ 1, subscript ], 8, 2 )
      ELSE IF subscript IN [ 9..16 ] THEN
        prompt_real( errorranges[ 2, subscript - 8 ], 8, 2 )
      ELSE IF subscript IN [ 17..24 ] THEN
        prompt_real( errorranges[ 1, subscript - 8 ], 8, 2 )
      ELSE IF subscript IN [ 25..32 ] THEN
        prompt_real( errorranges[ 2, subscript - 16 ], 8, 2 )
      ELSE CASE subscript OF
        33: prompt_real( errortime, 4, 0 );
        34: prompt_real( errorinterval, 4, 0 );
        35: prompt_real( currerror, 1, 0 );
        36: prompt_real( errorchan, 2, 0 );
        37: prompt_real( errorposition, 2, 0 );
        38: sho_bool( callok );
      END;      { if then elses, case... }
    END;        { with misc... }
END;    { show }

PROCEDURE sho_parms;

VAR
  subscript     :       integer;

BEGIN
  clrscrn;
  dispscreen( error_screen );
  goxy( 0, error_line + 2 );
  evon;
  prompt( 'Press <esc> to quit' );
  evoff;
  FOR subscript := 1 TO maxchan DO                { display the labels           }
    BEGIN
      IF subscript IN [ 1..8 ] THEN
        goxy( 5, subscript + 3 )
      ELSE
        goxy( 45, subscript - 5 );
      prompt( labels[ subscript ] );
    END;
  FOR subscript := 1 TO max_sub DO
    BEGIN
      move( subscript );
      show( subscript );
    END;        { for }
END;    { sho_parms }

PROCEDURE edit_parms;

VAR
  x, y,
  subscript     :       integer;

BEGIN
  subscript := 1;
  REPEAT
    WITH misc.errors DO
      BEGIN
        find_x_y( subscript, x, y );
        move( subscript );
        IF subscript IN [ 1..8 ] THEN
          errorranges[ 1, subscript ] := get_real( x, y,
              errorranges[ 1, subscript ], maxreal, minreal, last_char )
        ELSE IF subscript IN [ 9..16 ] THEN
          errorranges[ 2, subscript - 8 ] := get_real( x, y,
              errorranges[ 2, subscript - 8 ], maxreal, minreal, last_char )
        ELSE IF subscript IN [ 17..24 ] THEN
          errorranges[ 1, subscript - 8 ] := get_real( x, y,
              errorranges[ 1, subscript - 8 ], maxreal, minreal, last_char )
        ELSE IF subscript IN [ 25..32 ] THEN
          errorranges[ 2, subscript - 16 ] := get_real( x, y,
              errorranges[ 2, subscript - 16 ], maxreal, minreal, last_char )
        ELSE CASE subscript OF
          33: errortime := get_int( x, y, errortime, 5959, 0, last_char );
          34: errorinterval := get_int( x, y, errorinterval, 9999, 1,
                                                                last_char );
          35: currerror := get_int( x, y, currerror, chanerror, noerror,
                                                              last_char );
          36: errorchan := get_int( x, y, errorchan, maxchan, 1, last_char );
          37: errorposition := get_int( x, y, errorposition, maxposition,
                                                           1, last_char );
          38: get_bool( x, y, callok, last_char );
        END;         { case }
        move( subscript );
        show( subscript );
        IF ( subscript IN [ 8, 16, 24, 32 ] ) AND ( last_char = keydown ) THEN
          subscript := 33
        ELSE IF ( subscript IN [ 1, 9, 17, 25 ] ) AND
                                                 ( last_char = keyup ) THEN
          subscript := max_sub
        ELSE IF subscript IN [ 1..32 ] THEN
          update_subscript( last_char, subscript, max_sub, 8, 1 )
        ELSE
          update_subscript( last_char, subscript, max_sub, 1, 1 );
      END;      { with misc.errors... }
  UNTIL last_char = esc;
END;    { edit_parms }

BEGIN   { segment error_parm }
  IF executed( load_parms( vol, mirec ) ) THEN
    BEGIN
      sho_parms;
      edit_parms;
    END;
  perform( update_parms( vol, mirec ) );
END;    { segment error_parm }


segment PROCEDURE chan_parm;
{ edit the channels section of the miscellaneous program values record }

CONST
  max_sub       =       16;
  chan_screen   =       6;

VAR
  last_char     :       char;
  temp_set      :       setofchar;

PROCEDURE find_x_y( subscript: integer; VAR x, y: integer );

BEGIN
  IF subscript IN [ 1..8 ] THEN
    BEGIN
      x := 15;
      y := subscript + 2;
    END
  ELSE
    BEGIN
      x := 48;
      y := subscript - 6;
    END;
END;    { find_x_y }

PROCEDURE move( subscript: integer );

VAR
  x, y  :       integer;

BEGIN
  find_x_y( subscript, x, y );
  goxy( x, y );
END;    { move }

PROCEDURE show( subscript: integer );
```

```
BEGIN
  sho_voltrange( misc.channels[ subscript ] );
END;    { show }

PROCEDURE sho_parms;

VAR
  subscript     :       integer;

BEGIN
  clrscrn;
  dispscreen( chan_screen );
  goxy( 0, error_line + 2 );
  evon;
  prompt( 'Press <esc> to quit' );
  evoff;
  FOR subscript := 1 TO max_sub DO
    BEGIN
      IF subscript IN [ 1..8 ] THEN
        goxy( 3, subscript + 2 )
      ELSE
        goxy( 36, subscript - 6 );
      prompt( labels[ subscript ] );
    END;
  FOR subscript := 1 TO max_sub DO
    BEGIN
      move( subscript );
      show( subscript );
    END;        { for }
END;    { sho_parms }

PROCEDURE edit_parms;

VAR
  x, y,
  subscript     :       integer;

BEGIN
  subscript := 1;
  REPEAT
    find_x_y( subscript, x, y );
    move( subscript );
    get_voltrange( x, y, misc.channels[ subscript ],
                                     misc.channels[ subscript ], last_char );
    move( subscript );
    show( subscript );
    update_subscript( last_char, subscript, max_sub, 1, 1 );
  UNTIL last_char = esc;
END;    { edit_parms }

BEGIN   { segment chan_parm }
  IF executed( load_parms( vol, mirec ) ) THEN
    BEGIN
      sho_parms;
      edit_parms;
    END;
  perform( update_parms( vol, mirec ) );
END;    { segment chan_parm }


segment PROCEDURE trans_parm;
{ edit the transpiration rate section of the cart info record }

CONST
  max_sub       =       18;
  trans_screen  =       0;      { dummy screen number }
  VAR
    last_char     :       char;
    temp_set      :       setofchar;
    position      :       integer;

  PROCEDURE find_x_y( subscript: integer; VAR x, y: integer );

  BEGIN
  END;    { find_x_y }

  PROCEDURE move( subscript: integer );

  VAR
    x, y  :       integer;

  BEGIN
    find_x_y( subscript, x, y );
    goxy( x, y );
  END;    { move }

  PROCEDURE show( subscript: integer );

  BEGIN
  END;    { show }

  PROCEDURE sho_parms;

  VAR
    subscript     :       integer;

  BEGIN
    clrscrn;
    dispscreen( trans_screen );
    goxy( 0, error_line + 2 );
    evon;
    prompt( 'Press <esc> to quit' );
    evoff;
    FOR subscript := 1 TO max_sub DO
      BEGIN
        move( subscript );
        show( subscript );
      END;        { for }
  END;    { sho_parms }

  PROCEDURE edit_parms;

  VAR
    subscript     :       integer;

  BEGIN
    (* subscript := 1;
    REPEAT
      move( subscript );
      CASE subscript OF
      END:        { case }
      move( subscript );
      show( subscript );
      update_subscript( last_char, subscript, max_sub, 1, 1 );
    UNTIL last_char = esc; *)
  END;    { edit_parms }

BEGIN   { segment trans_parm }
  (* position := 1;
  clrscrn;
  promptln( 'Edit the stored transpiration values for one position' );
  REPEAT
    get_position( position );
    IF last_char <> esc THEN
```

```
      IF executed( load_parms( vol, position ) ) THEN        { loads cart var    }
        BEGIN
          sho_parms( position );
          edit_parms;
          perform( update_parms( vol, position ) );
          last_char := space;
        END;    { if last_char, if executed... }
  UNTIL last_char = esc; *)
END;    { segment trans_parm }


segment PROCEDURE std_parm;
{ edit the gasarray section of the current values record }

CONST
  max_sub       =       80;
  std_screen    =       0;      { dummy screen }

VAR
  last_char     :       char;
  temp_set      :       setofchar;

PROCEDURE find_x_y( subscript: integer; VAR x, y: integer );

BEGIN
END;    { find_x_y }

PROCEDURE move( subscript: integer );

VAR
  x, y  :       integer;

BEGIN
  find_x_y( subscript, x, y );
  goxy( x, y );
END;    { move }

PROCEDURE show( subscript: integer );

BEGIN
END;    { show }

PROCEDURE sho_parms;

VAR
  subscript     :       integer;

BEGIN
  clrscrn;
  dispscreen( std_screen );
  goxy( 0, error_line + 2 );
  evon;
  prompt( 'Press <esc> to quit' );
  evoff;
  FOR subscript := 1 TO max_sub DO
    BEGIN
      move( subscript );
      show( subscript );
    END;        { for }
END;    { sho_parms }

PROCEDURE edit_parms;

VAR
  subscript     :       integer;

BEGIN
  (* subscript := 1;
    REPEAT
      move( subscript );
      CASE subscript OF
      END;          { case }
      move( subscript );
      show( subscript );
      update_subscript( last_char, subscript, max_sub, 1, 1 );
    UNTIL last_char = esc; *)
  END;    { edit_parms }

  BEGIN   { std_parm }
    (* IF executed( load_parms( vol, curec ) ) THEN
      BEGIN
        sho_parms;
        edit_parms;
      END;
    perform( update_parms( vol, curec ) ); *)
  END;    { std_parm }
```

```
{$C StatPro WorkFile Select/Merge Program by Brian J. Martin, September 1984. }
{
  This program is designed to back up the workfile records on the Corvus onto
  floppy disks.  The user can choose to transfer all or some of the records.
  Selections may be based upon the date, time, position and/or cart fields of
  the workfile.  A numerical range of records to be transferred may also be
  specified.  The user may append to a previously created workfile, or start
  a new one.  The default transfer is all the workfile records on the source
  volume #11 to destination volume #9, creating a new workfile.  Apple Pascal
  formatted disks must be used, but they need not be blank.  If the disk con-
  tains one or more files, the program will ask for confirmation before writ-
  ing over the disk.  The program writes a DEFAULTS file on the disk, as well
  as the WKEYFILE, WLABELS, and WORKFILE files.  Record 0 of the source, the
  defaults record, is copied onto every new destination disk.
}
{
8-Apr-85 -mm- upgrade to Apple Pascal 1.2
11-Jun-85 -mm- small fix to allow chaining to a calling program
}
{$S+}
program SelectMerge;

uses
  CHAINSTUFF,
  CONTROLSTUFF,
  MICROMODEM,
  KYBDSTUFF;



const
  FloppyMaxRec    = 940;{ Maximum # of workfile records that fit on a floppy }

  KeyName         = 'WKEYFILE' ;
  WorkName        = 'WORKFILE' ;

  {$I CONTWRK:CONSTS.TEXT }


type
  Boundary      = record         { Used for date and time boundaries }
                    Low,
                    High: real;
                  end; { Boundary }

  RecRange      = record         { Used for low and high record # boundaries }
                    Low,
                    High: integer;
                  end; { RecRange }

  {$I CONTWRK:TYPES.TEXT }

  PosArray        = array[ 0..MaxPosition ] of boolean;

  CartArray       = array[ 0..MaxCart ] of boolean;

  WriteJustify    = (L, C, R);  { For Left-, Center- and Right-justifications }


var
  SourceKey,
  DestKey        : file of KeyRec;

  SourceWork,
  DestWork       : file of WorkRec;
```

```
    PermDate,
    TempDate,
    PermTime,
    TempTime        : Boundary;

    PermRec,
    TempRec         : RecRange;

    Position        : PosArray;

    Cart            : CartArray;

    LastRecNum,
    SourceDrive,
    DestDrive,
    OldSource       : integer;

    SourceVolume,            { concatenation of '#', the source volume #, and ':' }
    DestVolume,              {       "        "   , the destination #,     "    "  }
    OldName         : string;      { original name of the source volume }
    CtrlR,                   { used to return to Pascal instead of Control Program }
    UserInput       : char;  { used for getting user input from the keyboard }
    AllPositions,            { true if all positions are selected }
    AllCarts,                { true if all carts are selected }
    FirstExecution,          { true on first time through the program }
    Appending       : boolean;     { true if appending the workfile to an old one }
    to_program      : string;      { program to return to when finished }


procedure InitGlobals( var CtrlR: char; var Condition: CondType;
                       var FirstExecution, Appending: boolean;
                       var SourceDrive, DestDrive: integer;
                       var OldName: string );

begin { procedure InitGlobals }
  CtrlR := chr(18);
  Condition := Continue;
  FirstExecution := true;
  Appending := false;
  SourceDrive := 11;
  DestDrive := 9;
  OldName := 'ZZXZXZZ';
end; { procedure InitGlobals }


procedure InitStrings( SourceDrive, DestDrive: integer;
                       var SourceVolume, DestVolume: string );

begin { procedure InitStrings }
  str( SourceDrive, SourceVolume );
  SourceVolume := concat( '#', SourceVolume, ':' );
  str( DestDrive, DestVolume );
  DestVolume := concat( '#', DestVolume, ':' );
end; { procedure InitStrings }


procedure InitPositions( var Position: PosArray; var AllPositions: boolean;
                         Value: boolean );
var
  Loop: integer;

begin { procedure InitPositions }
```

```
  for Loop := 0 to MaxPosition do
    Position[ Loop ] := Value;
  AllPosition := Value ;
end; { procedure InitPositions }


procedure InitCarts( var Cart: CartArray; var AllCarts: boolean;
                     Value: boolean );

var
  Loop: integer;

begin { procedure InitCarts }
  for Loop := 0 to MaxCart do
    Cart[ Loop ] := Value;
  AllCarts := Value ;
end; { procedure InitCarts }


procedure Justify( YCoordinate: integer; How: WriteJustify; Message: string );

const
  ScreenWidth = 80;

var
  XCoordinate,
  MsgLength: integer;

begin { procedure Justify }
  MsgLength := length( Message );
  case How of
    L: XCoordinate := 0;
    C: XCoordinate := (ScreenWidth - MsgLength) div 2;
    R: XCoordinate := ScreenWidth - MsgLength;
  end; { case }
  gotoxy( XCoordinate, YCoordinate );
  write( Message );
end; { procedure Justify }


procedure CannotOpen( ErrorNum: integer; FileName: string ); forward;


procedure GetDirectory( VolumeNumber: integer; var Dir: DirPtr );

begin { procedure GetDirectory }
  new( Dir );
  {$I-}
  unitread( VolumeNumber, Dir^, sizeof( Directory ), 2 );
  if IOResult <> 0 then
    CannotOpen( IOResult, 'DIRECTORY' );
  {$I+}
end; { procedure GetDirectory }


procedure Rename( VolumeNumber: integer; NewName: VID; SaveOldName: boolean );

var
  Dir: DirPtr;
  Heap: ^integer;

begin { procedure Rename }
  mark( Heap );
```

```
    if NewName <> 'ZZXZXZZ' then begin
      GetDirectory( VolumeNumber, Dir );
      with Dir^[ 0 ] do begin
        if SaveOldName then
          OldName := DVID;
          if DVID <> NewName then begin
            DVID := NewName;
            {$I-}
            unitwrite( VolumeNumber, Dir^, sizeof( Directory ), 2 );
            if IOResult <> 0 then
              CannotOpen( IOResult, 'DIRECTORY' );
            {$I+}
          end; { if }
      end; { with }
    end; { if }
    release( Heap );
  end; { procedure Rename }



procedure CannotOpen{( ErrorNum: integer; FileName: string )};

var
  UserInput: char;

begin { procedure CannotOpen }
  writeln( Bell );
  page( output );
  gotoxy( 0, 10 );
  writeln( 'Error ', ErrorNum, ' occurred trying to open file ', FileName );
  Justify(12, L, 'Program aborted.  Press <ESC> to return to Control Program.');
  UserInput := ReadAChar( ' ', [], false );
  Rename( SourceDrive, OldName, false );
  setchain( '%CONTROL.CODE' );
  exit( program );
end; { procedure CannotOpen }



procedure GetPerms( SourceDrive: integer; SourceVolume: string;
                    var PermDate, TempDate, PermTime, TempTime: Boundary;
                    var PermRec, TempRec: RecRange );

begin { procedure GetPerms }
  {$I-}
  reset( SourceKey, concat(SourceVolume, KeyName) );
  if IOResult <> 0 then
    if FirstExecution and (UserInput in ['C', 'c']) then begin
      FirstExecution := false;
      close( SourceKey, lock );
      exit( GetPerms );
    end { if }
    else
      CannotOpen( IOResult, concat(SourceVolume, KeyName) );
  {$I+}
  PermRec.High := SourceKey^.RTotal - 1;
  PermRec.Low := 1;
  close( SourceKey, lock );
  {$I-}
  reset( SourceWork, concat(SourceVolume, WorkName) );
  if IOResult <> 0 then
    CannotOpen( IOResult, concat(SourceVolume, WorkName) );
  {$I+}
  seek( SourceWork, PermRec.Low );
  get( SourceWork );
  PermDate.Low := SourceWork^.Data[ DateField ];
  seek( SourceWork, PermRec.High );
  get( SourceWork );
```

```
    PermDate.High := SourceWork^.Data[ DateField ];
    close( SourceWork, lock );
    PermTime.Low := 0.0; { Lowest valid 24 hour time }
    PermTime.High := 23.5959; { Highest valid 24 hour time }
    TempDate := PermDate;
    TempTime := PermTime;
    TempRec := PermRec;
  end; { procedure GetPerms }



  procedure GetNewSource( var SourceDrive: integer );

  var
    UserInput: char;

  begin { procedure GetNewSource }
    writeln( Bell );
    page( output );
    Justify( 9, L, 'The default volume [#' );
    write( SourceDrive, '] does not contain the WKEYFILE.' );
    Justify( 11, L, 'Please enter the number of the new source drive-->' );
    gotoxy( 50, 11 );
    SourceDrive := GetInteger( SourceDrive, 12, 4 );
    Justify( 13, L, 'Press <RETURN> to continue, <ESC> to abort.');
    repeat
      gotoxy( 43, 13 );
      UserInput := ReadAChar( ' ', [], false );
    until (Condition = Abort) or (Condition = Same);
    if Condition = Abort then begin
      setchain( '%CONTROL.CODE' );
      exit( program );
    end; {if }
    page( output );
    gotoxy( 0, 11 );
    write( 'Reading/writing drive #', SourceDrive, '....' );
  end; { procedure GetNewSource }

{$I MISCWRK:SELECT.INC.TEXT }

procedure DoTransfer( SourceDrive, DestDrive: integer; var LastRecNum: integer;
                      SourceVolume, DestVolume: string; var Appending: boolean;
                      var TempRec: RecRange );

type
  DataArray = array[ 0..MaxWrkField ] of real;

var
  NumRecs,
  RecordNum: integer;



  function Legal( var Data: DataArray ): boolean;

  var
    TempTrunc: integer;
    Good: boolean;

  begin { function Legal }
    Good := true;
    if Data[ DateField ] = TempDate.Low then begin
      if Data[ TimeField ] < TempTime.Low then
        Good := false;
    end { if }
    else if Data[ DateField ] = TempDate.High then begin
      if Data[ TimeField ] > TempTime.High then
        Good := false;
    end { if }
    else if (Data[ DateField ] < TempDate.Low) or
            (Data[ DateField ] > TempDate.High) then
      Good := false;
    if Good then begin
      TempTrunc := trunc(Data[ PosCartField ]);
      if (not(Position[ TempTrunc ])) or
         (not(Cart[ round((Data[ PosCartField ] - TempTrunc) * 100) ])) then
        Good := false;
    end; { if }
    Legal := Good;
  end; { function Legal }


  procedure CloseDestination( TotalNumRecs: integer );

  begin { procedure CloseDestination }
    page( output );
    gotoxy( 0, 11 );
    write( 'Closing drive #', DestDrive, '....' );
    close( DestWork, lock );
    {$I-}
    reset( DestKey, concat(DestVolume, KeyName) );
    if IOResult <> 0 then
      CannotOpen( IOResult, concat(DestVolume, KeyName) );
    {$I+}
    DestKey^.RTotal := TotalNumRecs;
    seek( DestKey, 0 );
    put( DestKey );
    close( DestKey, lock );
    Rename( DestDrive, 'WORKFIL', false );
  end; { procedure CloseDestination }


  procedure PutZeroGetNext( Appending: boolean; var NumRecs: integer;
                            LastRecNum, NextRecord: integer );

  begin { procedure PutZeroGetNext }
    if not Appending then begin { record 0 goes on every disk }
      seek( SourceWork, 0 );
      get( SourceWork );
      DestWork^ := SourceWork^;
      put( DestWork );
      NumRecs := 1;
    end
    else
      NumRecs := LastRecNum + 1;
    seek( SourceWork, NextRecord );
  end; { procedure PutZeroGetNext }


  procedure DiskFull( var NumRecs, LastRecNum: integer; var TempRec: RecRange;
                      var Appending: boolean );
  var
    UserInput: char;

  begin { procedure DiskFull }
    CloseDestination( FloppyMaxRec );
    if RecordNum <> TempRec.High then begin
      writeln( Bell );
      page( output );
      Justify( 10, L, 'Destination disk is now full.  Insert a new disk' );
      Justify( 12, L, 'and press <RETURN> to continue, <ESC> to abort.' );
      repeat
```

```
          gotoxy( 47, 12 );
          UserInput := ReadAChar( ' ', [], false );
        until (Condition = Abort) or (Condition = Same);
        if Condition = Abort then begin
          Justify( 14, L, 'Returning to Control Program....' );
          Rename( SourceDrive, OldName, false );
          setchain( '%CONTROL.CODE');
          exit( program );
        end; { if }
        if Appending then begin
          Justify( 14, L, 'Will you STILL be appending to a previously ' );
          write( 'created workfile (Y/N)?');
          UserInput := GetChar( ['Y', 'y', 'N', 'n'], true );
          Appending := UserInput in ['Y', 'y'];
        end; { if }
        OpenDestination( SourceVolume, DestVolume, DestDrive,
                         LastRecNum, Appending );
        page( output );
        Justify( 10, L, 'Transferring workfile....' );
        PutZeroGetNext( Appending, NumRecs, LastRecNum, RecordNum + 1 );
      end; { if }
    end; { procedure DiskFull }


  begin { procedure DoTransfer }
    page( output );
    Justify( 10, L, 'Transferring workfile....' );
    {$I-}
    reset( SourceWork, concat(SourceVolume, WorkName) );
    if IOResult <> 0 then
      CannotOpen( IOResult, concat(SourceVolume, WorkName) );
    {$I+}
    PutZeroGetNext( Appending, NumRecs, LastRecNum, TempRec.Low );
    RecordNum := TempRec.Low;
    get( SourceWork );
    while RecordNum <= TempRec.High do begin
      if Legal( SourceWork^.Data ) then begin
        DestWork^ := SourceWork^;
        put( DestWork );
        NumRecs := NumRecs + 1;
        gotoxy( 0, 12 );
        write( 'Number of records on destination disk: ', NumRecs );
      end; { if }
      if NumRecs = FloppyMaxRec then
        DiskFull( NumRecs, LastRecNum, TempRec, Appending );
      get( SourceWork );
      RecordNum := RecordNum + 1;
    end; { while }
    close( SourceWork, lock );
    Rename( SourceDrive, OldName, false );
    if NumRecs <> FloppyMaxRec then
      CloseDestination( NumRecs );
end; { procedure DoTransfer }


begin { program SelectMerge }
  getcval( to_program );
  InitGlobals( CtrlR, Condition, FirstExecution, Appending,
               SourceDrive, DestDrive, OldName );
  InitStrings( SourceDrive, DestDrive, SourceVolume, DestVolume );
  repeat
    page( output );
    Justify( 0, C, '--- StatPro WorkFile Select/Merge Program Main Menu ---' );
    Justify( 4, L, 'Options:' );
    if FirstExecution then
      Justify( 7, L, 'T(ransfer workfile using default options' )
      else
        Justify( 7, L, 'T(ransfer workfile using existing options' );
      Justify( 9, L, 'C(hange existing transfer options' );
      Justify( 11, L, 'R(eturn to ' );
      if length( to_program ) > 0 then
        write( to_program )
      else
        write( 'Pascal operating system' );
      Justify( 14, L, 'Selection?' );
      UserInput := ReadAChar( 'R', ['T', 't', 'C', 'c', 'R', 'r', CtrlR], true );
      if FirstExecution and (UserInput in ['T', 't', 'C', 'c']) then begin
        page( output );
        gotoxy( 0, 11 );
        write( 'Reading/writing drive #', SourceDrive, '....' );
        InitPositions( Position, AllPositions, true );
        InitCarts( Cart, AllCarts, true );
        GetPerms( SourceDrive, SourceVolume, PermDate, TempDate,
                  PermTime, TempTime, PermRec, TempRec );
        if not FirstExecution then begin
          GetNewSource( SourceDrive );
          InitStrings( SourceDrive, DestDrive, SourceVolume, DestVolume );
          GetPerms( SourceDrive, SourceVolume, PermDate, TempDate,
                    PermTime, TempTime, PermRec, TempRec );
        end { if }
        else
          FirstExecution := false;
        OldSource := SourceDrive;
      end; { if }
      case UserInput of
        'T', 't'  : begin
                      Rename( SourceDrive, 'ZZZZZZZ', true );
                      OpenDestination( SourceVolume, DestVolume, DestDrive,
                                       LastRecNum, Appending );
                      DoTransfer( SourceDrive, DestDrive, LastRecNum,
                                  SourceVolume, DestVolume, Appending, TempRec );
                    end;
        'C', 'c'  : ChangeOptions( SourceDrive, DestDrive, OldSource,
                                   SourceVolume, DestVolume,
                                   PermDate, TempDate, PermTime, TempTime,
                                   PermRec, TempRec, Appending,
                                   AllPositions, AllCarts, Position, Cart );
      end; { case }
    until ( UserInput in [ 'r', 'R', CtrlR ] );
    if length( to_program ) > 0 then
      begin
        page( output );
        justify( 10, L, concat( 'Returning to ', to_program ) );
        setchain( to_program );
      end
    else
      justify( 10, L, 'Returning to Pascal Operating system' );
end. { program SelectMerge }
```

```
{
  SELECT.INC.TEXT is an include file for the StatPro WorkFile Select/Merge
  Program written by Brian J. Martin in September 1984.
}

procedure OpenDestination( SourceVolume, DestVolume: string; DestDrive: integer;
                           var LastRecNum: integer; Appending: boolean );

var
  UserInput: char;



  procedure PrepareDest( SourceVolume, DestVolume: string; DestDrive: integer );


    procedure DoDirectory( DestDrive: integer );

    var
      Loop: integer;
      UserInput: char;
      PokeDate: SetDate;
      Dir: DirPtr;
      Heap: ^integer;

    begin { procedure DoDirectory }
      mark( Heap );
      repeat
        GetDirectory( DestDrive, Dir );
        with Dir^[ 0 ] do begin
          if DNumFiles <> 0 then begin
            writeln( Bell );
            page( output );
            gotoxy( 0, 11 );
            write( 'Destroy ', DVID, ': (Y/N)?' );
            UserInput := ReadAChar( 'N', ['Y', 'y', 'N', 'n'], true );
            page( output );
            if UserInput in ['Y', 'y'] then begin { zero out directory }
              gotoxy( 0, 11 );
              write( 'Preparing drive #', DestDrive, '....' );
              DVID := 'WORKFIL';
              for Loop := 1 to DNumFiles do
                Dir^[ Loop ].DFID := '';
              DNumFiles := 0;
              PokeDate.DateLoc := TheDate;
              DLastBoot := PokeDate.DatePtr^;
              {$I-}
              unitwrite( DestDrive, Dir^, sizeof( Directory ), 2 );
              if IOResult <> 0 then
                CannotOpen( IOResult, 'DIRECTORY' );
              {$I+}
              Condition := Same;
            end { if }
            else begin { give 'em a chance to insert a blank }
              Justify( 11, L, 'Insert a blank disk and press <RETURN> to' );
              write( ' continue, <ESC> to abort.' );
              repeat
                gotoxy( 67, 11 );
                UserInput := ReadAChar( ' ', [], false );
              until (Condition = Abort) or (Condition = Same);
              if Condition = Abort then begin
                PromptAt( 13, 'Returning to Control Program....' );
                Rename( SourceDrive, OldName, false );
                setchain( '%CONTROL.CODE' );
                exit( program );
              end; { if }
          page( output );
          gotoxy( 0, 11 );
          write( 'Preparing drive #', DestDrive, '....' );
          Condition := Abort;
        end; { else }
      end { if }
      else
        Condition := Same;
    end; { with }
  until Condition = Same;
  release( Heap );
end; { procedure DoDirectory }


procedure DoDefaults( DestVolume: string );

const
  MaxDefault      =  70;
  Default         = -999.0;
  DefaultName     = 'DEFAULTS' ;

type
  Defaults = record
    Data: array[ 1..MaxDefault ] of real;
  end; { Defaults }

var
  Loop: integer;
  DummyDefault  : file of Defaults;

begin { procedure DoDefaults }
  {$I-}
  rewrite( DummyDefault, concat(DestVolume, DefaultName) );
  if IOResult <> 0 then
    CannotOpen( IOResult, concat(DestVolume, DefaultName) );
  {$I+}
  for Loop := 1 to MaxDefault do
    DummyDefault^.Data[ Loop ] := Default;
  put( DummyDefault );
  close( DummyDefault, lock );
end; { procedure DoDefaults }


procedure DoKeys( SourceVolume, DestVolume: string );

begin { procedure DoKeys }
  {$I-}
  reset( SourceKey, concat(SourceVolume, KeyName) );
  if IOResult <> 0 then
    CannotOpen( IOResult, concat(SourceVolume, KeyName) );
  rewrite( DestKey, concat(DestVolume, KeyName) );
  if IOResult <> 0 then
    CannotOpen( IOResult, concat(DestVolume, KeyName) );
  {$I+}
  DestKey^ := SourceKey^;
  put( DestKey );
  close( SourceKey, lock );
  close( DestKey, lock );
end; { procedure DoKeys }


procedure DoLabels( SourceVolume, DestVolume: string );

const
  LabelName       = 'WLABELS'  ;
```

```
    var
      SourceLabel,
      DestLabel     : file of LabelRec;

    begin { procedure DoLabels }
      {$I-}
      reset( SourceLabel, concat(SourceVolume, LabelName) );
      if IOResult <> 0 then
        if IOResult = 10 then { If labels file is not there, keep going }
          exit( DoLabels )
        else { Some other error other than file not found }
          CannotOpen( IOResult, concat(SourceVolume, LabelName) );
      rewrite( DestLabel, concat(DestVolume, LabelName) );
      if IOResult <> 0 then
        CannotOpen( IOResult, concat(DestVolume, LabelName) );
      {$I+}
      while not eof( SourceLabel ) do begin
        DestLabel^ := SourceLabel^;
        put( DestLabel );
        get( SourceLabel );
      end; { while }
      close( SourceLabel, lock );
      close( DestLabel, lock );
    end; { procedure DoLabels }



    procedure DoWorkFile( DestVolume: string );

    begin { procedure DoWorkFile }
      {$I-}
      rewrite( DestWork, concat(DestVolume, WorkName) );
      if IOResult <> 0 then
        CannotOpen( IOResult, concat(DestVolume, WorkName) );
      {$I+}
    end; { procedure DoWorkFile }



  begin { procedure PrepareDest }
    DoDirectory( DestDrive );
    DoDefaults( DestVolume );
    DoKeys( SourceVolume,DestVolume );
    DoLabels( SourceVolume, DestVolume );
    DoWorkFile( DestVolume );
  end; { procedure PrepareDest }


begin { procedure OpenDestination }
  page( output );
  gotoxy( 0, 11 );
  write( 'Preparing drive #', DestDrive, '....' );
  if Appending then begin
    {$I-}
    reset( DestWork, concat(DestVolume, WorkName) );
    if IOResult <> 0 then
      CannotOpen( IOResult, concat(DestVolume, WorkName) );
    reset( DestKey, concat(DestVolume, KeyName) );
    if IOResult <> 0 then
      CannotOpen( IOResult, concat(DestVolume, KeyName) );
    {$I+}
    LastRecNum := DestKey^.RTotal - 1;
    close( DestKey, lock );
    if LastRecNum < FloppyMaxRec - 1 then begin
      seek( DestWork, LastRecNum );
      get( DestWork );
```

```
        end
        else begin
          writeln( Bell );
          page( output );
          Justify( 10, L, 'Error--disk contains ' );
          writeln( LastRecNum + 1, ' records and cannot be appended to.' );
          Justify( 12, L, 'Program aborted.  Press <ESC> to return to Control' );
          write( ' Program.' );
          UserInput := ReadAChar( ' ', [], false );
          Rename( SourceDrive, OldName, false );
          setchain( '%CONTROL.CODE' );
          exit( program );
        end; { else }
      end { if Appending }
      else
        PrepareDest( SourceVolume, DestVolume, DestDrive );
    end; { procedure OpenDestination }


    procedure ChangeOptions( var SourceDrive, DestDrive, OldSource: integer;
                             var SourceVolume, DestVolume: string;
                             var PermDate, TempDate, PermTime, TempTime: Boundary;
                             var PermRec, TempRec: RecRange;
                             var Appending, AllPositions, AllCarts: boolean;
                             var Position: PosArray; var Cart: CartArray );
    var
      UserInput: char;


      procedure GetPositions( var Position: PosArray );

      var
        Loop,
        PosNum: integer;

      begin { procedure GetPositions }
        repeat
          page( output );
          Justify( 0, C, '--- Po[illegible]n Selection Menu ---' );
          if AllPositions then begin
            Justify( 4, L, 'All positions, [0..' );
            write( MaxPosition, '] are currently selected.' );
          end { if }
          else begin
            Justify( 4, L, 'The following positions are currently selected:' );
            gotoxy( 0, 6 );
            for Loop := 0 to MaxPosition do
              if Position[ Loop ] then
                write( Loop:4 );
          end; { else }
          gotoxy( 0, 8 );
          Justify( 8, L, 'Position # [0..' );
          write( MaxPosition, '], <+> select all positions, <ESC> accepts. ' );
          PosNum := GetInteger( 0, MaxPosition, 0 );
          if Condition = Same then begin
            if AllPositions then
              InitPositions( Position, AllPositions, false );
            Position[ PosNum ] := true;
          end { if }
          else if Condition = Continue then
            InitPositions( Positions, AllPositions, true );
        until Condition = Abort;
        Condition := Continue;
      end; { procedure GetPositions }
```

```
  procedure GetCarts( var Cart: CartArray );

  var
    Counter,
    Loop,
    CartNum: integer;

  begin { procedure GetCarts }
    repeat
      page( output );
      Justify( 0, C, '--- Cart Selection Menu ---' );
      if AllCarts then begin
        Justify( 4, L, 'All carts, [0..' );
        write( MaxCart, '] are currently selected.' )
      end { if }
      else begin
        Justify( 4, L, 'The following carts are currently selected:' );
        gotoxy( 0, 6 );
        Counter := 0;
        for Loop := 0 to MaxCart do begin
          if Cart[ Loop ] then begin
            write( Loop:4 );
            Counter := Counter + 1;
          end; { if }
          if Counter = 20 then begin
            Counter := 0;
            writeln;
          end; { if }
        end; { for }
      end; { else }
      Justify( 12, L, 'Cart # [0..' );
      write( MaxCart, '], <+> select all carts, <ESC> accepts. ' );
      CartNum := GetInteger( 0, MaxCart, 0 );
      if Condition = Same then begin
        if AllCarts then
          InitCarts( Cart, AllCarts, false );
        Cart[ CartNum ] := true;
      end { if }
      else if Condition = Continue then
        InitCarts( Cart, AllCarts, true );
    until Condition = Abort;
    Condition := Continue;
  end; { procedure GetCarts }


procedure GetOtherOptions( var SourceDrive, DestDrive, OldSource: integer;
                           var SourceVolume, DestVolume: string;
                           var PermDate, TempDate,
                               PermTime, TempTime: Boundary;
                           var PermRec, TempRec: RecRange;
                           var Appending: boolean );

var
  Line: integer;
  YesOrNo: char;


  procedure Move( Line: integer );

  begin { procedure Move }
   gotoxy( 27, Line );
  end; { procedure Move }
```

```
procedure WriteValues( Line: integer );

begin { procedure WriteValues }
  case Line of
     4: write( SourceDrive:2 );
     6: write( DestDrive:2 );
     8: write( TempRec.Low:5 );
    10: write( TempRec.High:5 );
    12: write( TempDate.Low:7:4 );
    14: write( TempDate.High:7:4 );
    16: write( TempTime.Low:7:4 );
    18: write( TempTime.High:7:4 );
    20: write( YesOrNo );
  end; { case }
end; { procedure WriteValues }



procedure DisplayScreen;

var
  Line: integer;

begin { procedure DisplayScreen }
  page( output );
  Justify( 0, C, '--- Other Options Menu ---' );
  gotoxy( 0, 4 );
  writeln( 'Source drive              [  ]' );
  writeln;
  writeln( 'Destination drive         [  ]' );
  writeln;
  writeln( 'Starting record number    [     ]' );
  writeln;
  writeln( 'Ending record number      [     ]' );
  writeln;
  writeln( 'Starting date             [       ]' );
  writeln;
  writeln( 'Ending date               [       ]' );
  writeln;
  writeln( 'Starting time             [       ]' );
  writeln;
  writeln( 'Ending time               [       ]' );
  writeln;
  writeln( 'Appending to old workfile [ ]' );
  Line := 2;
  while Line < 22 do begin
    Line := Line + 2;
    Move( Line );
    WriteValues( Line );
  end; { while }
  Justify(23, L, '<-- Move up, --> Move down, <ESC> to accept and return');
  write( ' to Change Options Menu' );
end; { procedure DisplayScreen }


procedure GetInput( Line: integer );

begin { procedure GetInput }
  case Line of
     4: SourceDrive := GetInteger( SourceDrive, 12, 4 );
     6: DestDrive := GetInteger( DestDrive, 12, 4 );
     8: TempRec.Low := GetInteger( TempRec.Low, PermRec.High, PermRec.Low );
    10: TempRec.High := GetInteger(TempRec.High, PermRec.High, PermRec.Low);
    12: TempDate.Low := GetFPNum(TempDate.Low, PermDate.High, PermDate.Low);
    14: TempDate.High := GetFPNum(TempDate.High,PermDate.High,PermDate.Low);
```

```
        16: TempTime.Low := GetFPNum(TempTime.Low, PermTime.High, PermTime.Low);
        18: TempTime.High := GetFPNum(TempTime.High,PermTime.High,PermTime.Low);
        20: begin
              YesOrNo := ReadAChar( YesOrNo,[ 'N','n','Y','y' ], true );
              Appending := YesOrNo in ['Y', 'y'];
            end;
      end; { case }
    end; { procedure GetInput }


  begin { procedure GetOtherOptions }
    if Appending then
      YesOrNo := 'Y'
    else
      YesOrNo := 'N';
    Condition := Continue;
    DisplayScreen;
    Line := 4;
    repeat
      Move( Line );
      GetInput( Line );
      Move( Line );
      WriteValues( Line );
      if Condition = Next then
        Line := Line + 2
      else if Condition = Previous then
        Line := Line - 2;
      if Line > 20 then
        Line := 4
      else if Line < 4 then
        Line := 20;
      if OldSource <> SourceDrive then begin
        page( output );
        Justify( 11, L, 'Reading new values from drive #' );
        write( SourceDrive, '....' );
        OldSource := SourceDrive;
        if SourceDrive = 4 then
          DestDrive := 5
        else if SourceDrive = 5 then
          DestDrive := 4
        else if SourceDrive = 9 then
          DestDrive := 11
        else
          DestDrive := 9;
        InitStrings( SourceDrive, DestDrive, SourceVolume, DestVolume );
        GetPerms( SourceDrive, SourceVolume, PermDate, TempDate,
                  PermTime, TempTime, PermRec, TempRec );
        DisplayScreen;
      end; { if }
    until Condition = Abort;
    InitStrings( SourceDrive, DestDrive, SourceVolume, DestVolume );
    Condition := Continue;
  end; { procedure GetOtherOptions }


begin { procedure ChangeOptions }
  repeat
    page( output );
    Justify( 0, C, '--- Change Options Menu ---' );
    Justify( 4, L, 'Options:' );
    Justify( 7, L, 'P(osition Selection' );
    Justify( 9, L, 'C(art Selection' );
    Justify( 11, L, 'O(ther Options [Drives, Dates, Times, and Record #''s]' );
    Justify( 13, L, 'R(eturn to main menu' );
    Justify( 16, L, 'Selection?' );
    UserInput := ReadAChar('R', ['P', 'p', 'C', 'c', 'O', 'o', 'R', 'r'], true);
    case UserInput of
      'P', 'p': GetPositions( Position );
      'C', 'c': GetCarts( Cart );
      'O', 'o': GetOtherOptions( SourceDrive, DestDrive, OldSource,
                                 SourceVolume, DestVolume,
                                 PermDate, TempDate, PermTime, TempTime,
                                 PermRec, TempRec, Appending );
      'R', 'r': Condition := Continue;
    end; { case }
  until (UserInput = 'R') or (UserInput = 'r');
end; { procedure ChangeOptions }
```

APPENDIX 3

PLANT EXPOSURE CHAMBER CONSTRUCTION DRAWINGS AND PARTS LIST

by

Thomas Pfleeger

Toxic and Hazardous Materials Branch
Corvallis Environmental Research Laboratory
200 S.W. 35th Street
Corvallis, Oregon 97333

CORVALLIS ENVIRONMENTAL RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CORVALLIS, OREGON 97333

DISCLAIMER

The information in this document has been funded wholly or in part by the U.S. Environmental Protection Agency. It has been subjected to the Agency's peer and administrative reviews and has been approved for publication as an EPA document. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

CONTENTS

Page
Figures 3-iv
Introduction 3-1
Exposure Chamber Drawings 3-2
Plant Exposure Chamber Parts List 3-9
    Air System 3-9
    Cart Frame 3-11
    Hydroponic System 3-12
Manufacturers and Suppliers 3-16

FIGURES

Number Page

3-1 Plant Exposure Chamber Exploded View 3-2

3-2 Plant Exposure Chamber Cart 3-3

3-3 Cart Frame Detail 3-3

3-4 Hydroponic System Detail 3-4

3-5 Hydroponic System Manifold Exploded View 3-5

3-6 Hydroponic System Solution Tank Exploded View 3-5

3-7 Condenser Unit Exploded View 3-6

3-8 Probe Pot Exploded View 3-7

3-9 Electronic Module Detail 3-8

3-10 Front Panel Detail 3-8

INTRODUCTION

The figures show construction details for various components of the plant exposure chambers. If more detail on construction methods is desired it can be supplied by the authors.

The parts list is a tabulation of the items used in the assembly of an exposure chamber. The listing is organized according to where a particular item is used on a chamber (e.g., frame, hydroponic system, or air system). This list does not contain any of the components of the electronic module. These are found in Appendix 4.

The manufacturer and supplier list is a directory of the names and addresses of those companies that can supply the items on the parts list (or similar items). Those are not the only companies supplying these items but are the ones we found the most convenient to Corvallis, Oregon.

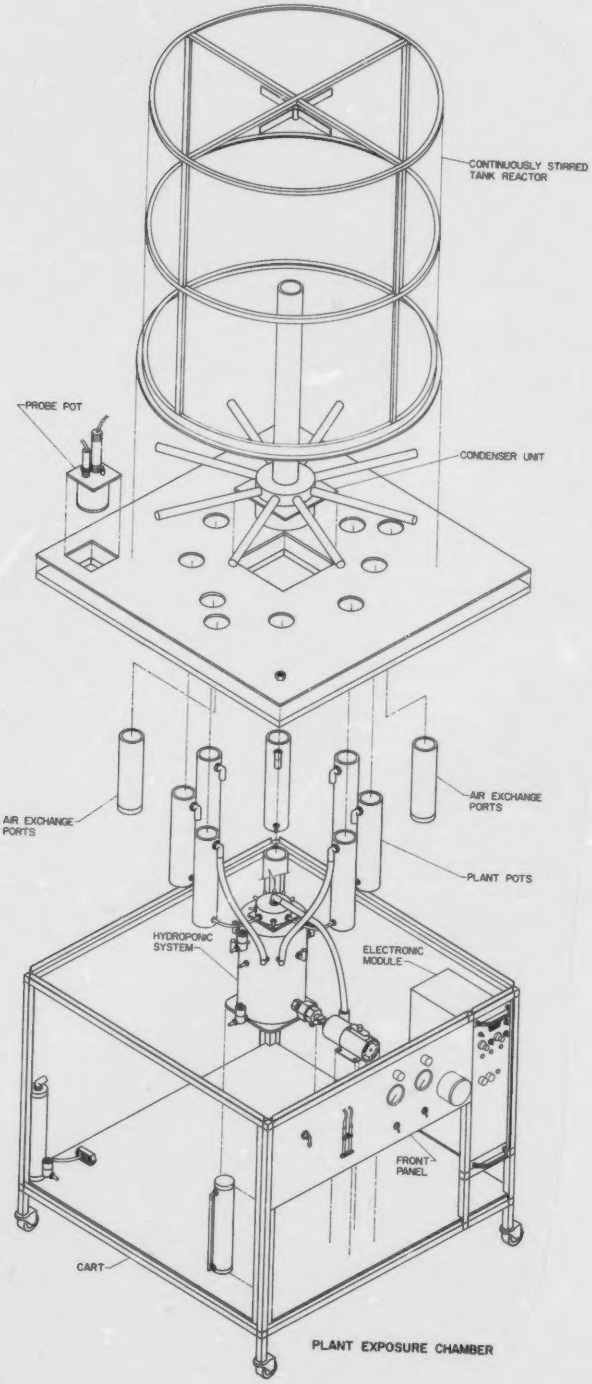

CONTINUOUSLY STIRRED TANK REACTOR
PROBE POT
CONDENSER UNIT
AIR EXCHANGE PORTS
AIR EXCHANGE PORTS
PLANT POTS
HYDROPONIC SYSTEM
ELECTRONIC MODULE
FRONT PANEL
CART
PLANT EXPOSURE CHAMBER

Figure 3-1. Plant exposure chamber exploded view.

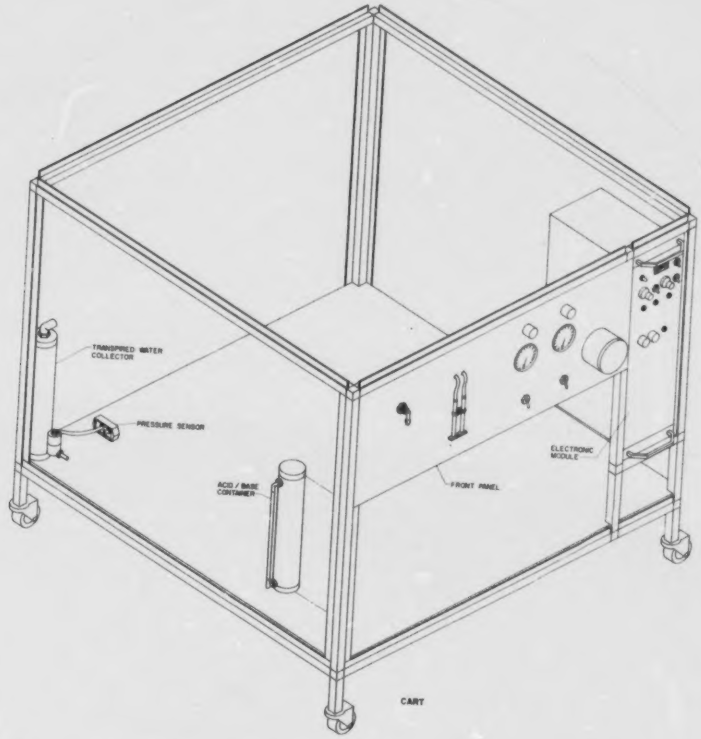

TRANSPIRED WATER COLLECTOR
PRESSURE SENSOR
ACID / BASE CONTAINER
FRONT PANEL
ELECTRONIC MODULE
CART

Figure 3-2. Plant exposure chamber cart. Note the locations of the transpired water collector, acid/base container, front instrument panel, and electronic module.

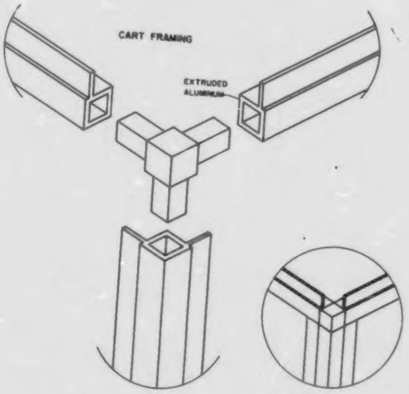

CART FRAMING
EXTRUDED ALUMINUM

Figure 3-3. Cart frame detail. The framing materials are extruded aluminum pieces, the corner assemblies are cast aluminum. The corner assembly is friction fitted with locking clips to the framing pieces.

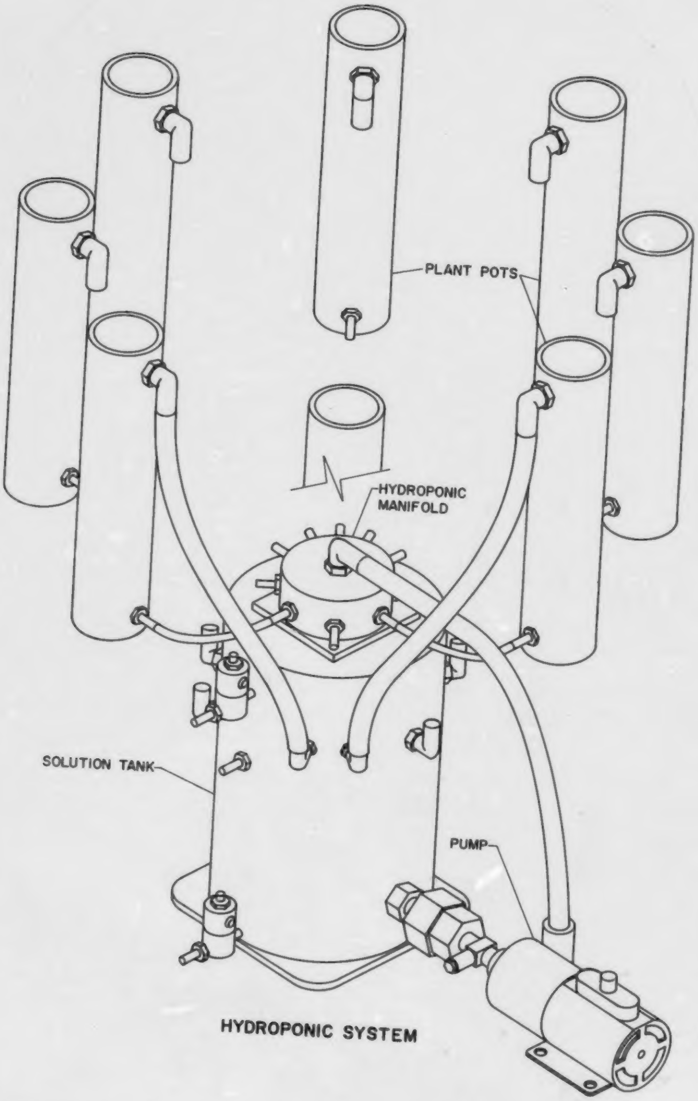


Figure 3-4. Hydroponic system detail. The flow of solution in the system is as follows. The nutrient solution is pumped from the solution tank to the hydroponic manifold. Solution enters the bottom of each plant pot from the manifold. (Not shown are the solution lines from the manifold to the sampling port and to the probe pot.) The solution returns to the solution tank via a drain line from the top of each plant pot. A union fitting connects the pump to the solution tank for ease in pump replacement. Adjacent to the union is another fitting used to add oxygen to the solution. All materials in contact with the nutrient solution are composed of polyethylene, teflon, or stainless steel.

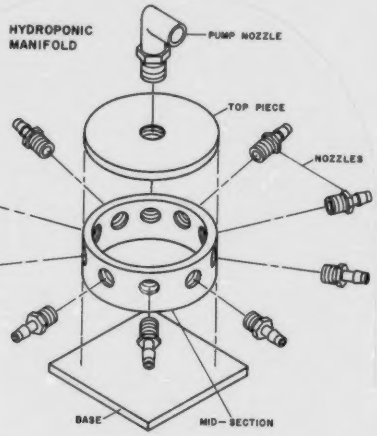

HYDROPONIC MANIFOLD
PUMP NOZZLE
TOP PIECE
NOZZLES
BASE
MID-SECTION

Figure 3-5. Hydroponic manifold exploded view. The midsection is a 1" piece of 3" ID polyethylene pipe (Driscopipe). The midsection is heat fused to the base and top piece which are made from a 1/4" thick piece of polyethylene flat stock. Sites for nozzles are drilled and tapped.

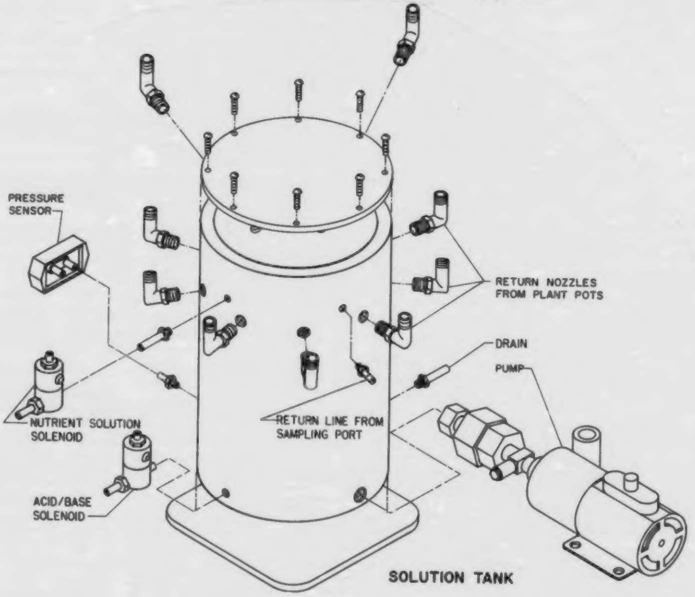

PRESSURE SENSOR
RETURN NOZZLES FROM PLANT POTS
DRAIN
PUMP
NUTRIENT SOLUTION SOLENOID
RETURN LINE FROM SAMPLING PORT
ACID/BASE SOLENOID
SOLUTION TANK

Figure 3-6. Hydroponic system solution tank sxploded view. The bottom piece of 1/4" thick polyethylene flat stock is heat fused to a 12" section of 8" ID polyethylene pipe (Driscopipe). All other parts are fitted by drilling and tapping holes into the pipe. A pressure sensor is used to determine the solution volume. The nutrient solution solenoid is opened when additional solution is needed. The acid/base solenoid is opened when a pH adjustment is needed. Not shown is a float valve which acts as a safety backup to close the nutrient solution solenoid in case of a pressure sensor failure and the solution level rises too high.

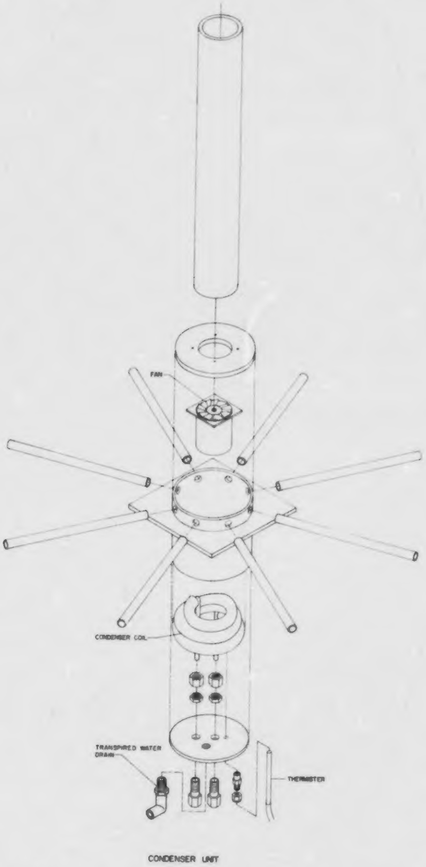
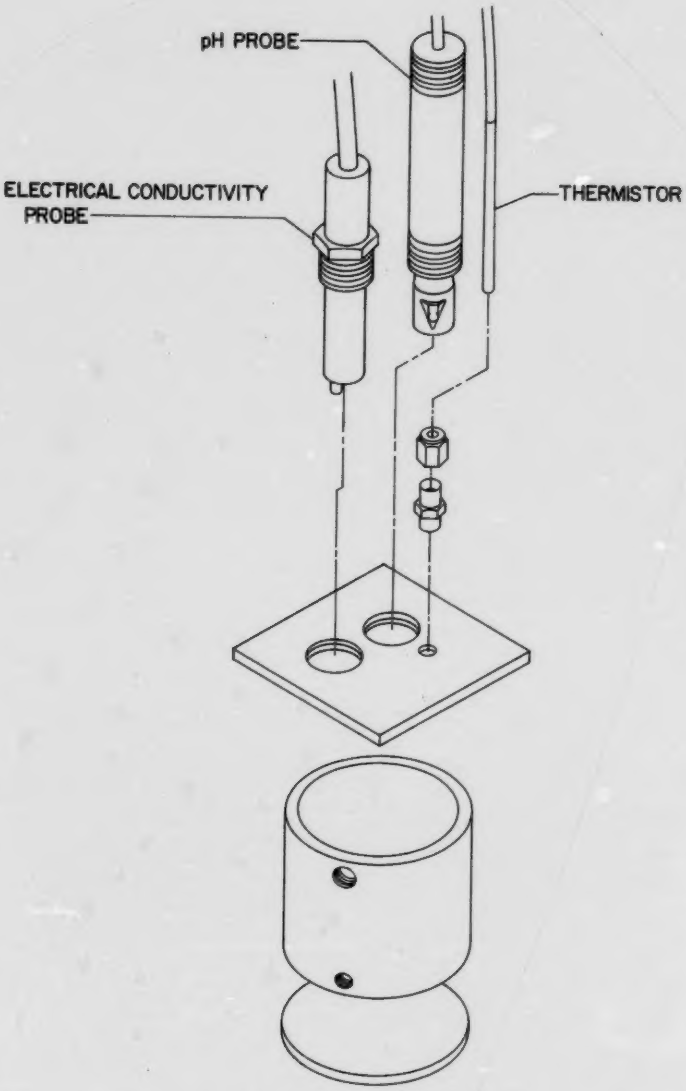


Figure 3-7. Condenser unit exploded view. Note the placement of the condenser coil in relationship to the fan. The unit is made of a 4" section of 5" ID polyethylene pipe (Driscopipe) heat fused to a piece of 1/4" thick polyethylene flat stock on top with a similar piece of flat stock bolted to the bottom. A hole the size of the fan is drilled in the top and the fan is installed underneath it. A piece of 1/2" ID copper tubing (a copper finned radiator) is bent to fit around the fan and plumbed thru the bottom of the unit. A 7.5 by 7.5" square collar of 1/4" polyethylene flat stock fits around the condenser unit and supports it on the chamber table. The collar is attached to the unit by four metal pins. Eight wands made from polyethylene tubing are spaced equally around the unit top such that they lay on the chamber top. The central tower is a 25" piece of 2" ID polyethylene pipe (Driscopipe) that is insulated and then covered with teflon film. The tower is held in place by a friction fit inside the fan hole opening. The condenser coil (the finned radiator) has cold water pumped through it. The temperature of this water is measured by a thermistor. Water condensed on the coil is drained to the transpired water collector. The fan blows air out of the tower and pulls air in the eight polyethylene wands across the chilled coil. The chamber humidity is controlled by the temperature of the condenser coil and the air speed.



PROBE POT

Figure 3-8. Probe pot exploded view. The probe pot contains sensors for pH, electrical conductivity and temperature. It is made by heat fusing a piece of 1/4" thick polyethylene flat stock to a 3.25" long piece of 3" ID polyethylene pipe (Driscopipe). The top is a 4" square of 1/4" thick polyethylene flat stock which is bolted to the other end of the pipe. Holes for the probes are drilled and tapped to the appropriate size for the desired probe in the top piece. Holes are drilled and tapped in the top and bottom of the pipe for the nutrient supply and return fittings. The top square piece suspends the probe pot from the table surface.

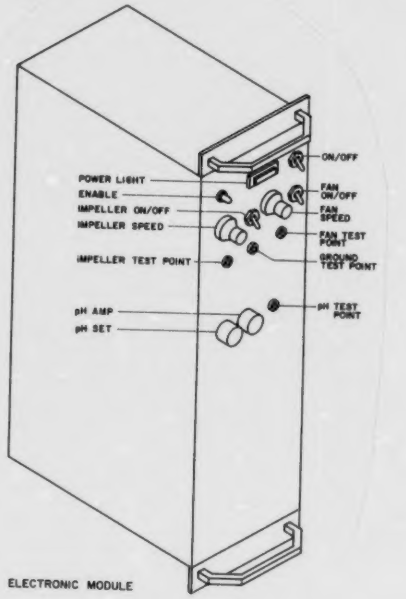

ON/OFF
POWER LIGHT
ENABLE
FAN ON/OFF
IMPELLER ON/OFF
FAN SPEED
IMPELLER SPEED
FAN TEST POINT
IMPELLER TEST POINT
GROUND TEST POINT
pH AMP
pH TEST POINT
pH SET
ELECTRONIC MODULE

Figure 3-9. Electronic module detail. The electronic module is interchangeable between the different chambers and also between different chamber types (soil and solution). It has external speed controls for the condenser fan and impeller. It also has the calibration controls for setting pH. Internally it contains the electronics for controlling the chamber and for sending/receiving signals from the computer. Electronic details are included in Appendix 4.

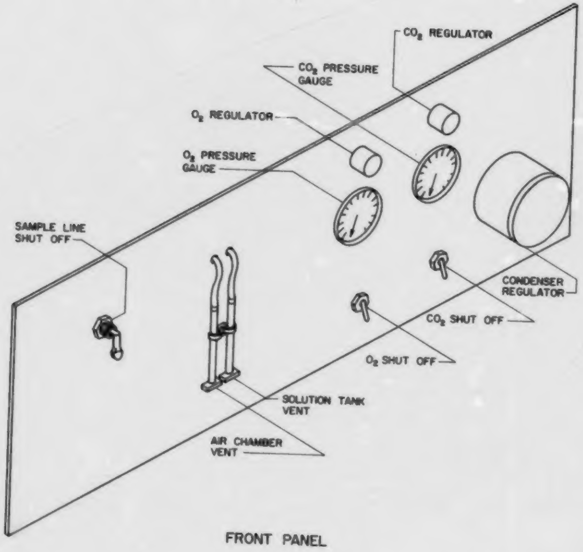

CO₂ REGULATOR
CO₂ PRESSURE GAUGE
O₂ REGULATOR
O₂ PRESSURE GAUGE
SAMPLE LINE SHUT OFF
CONDENSER REGULATOR
CO₂ SHUT OFF
O₂ SHUT OFF
SOLUTION TANK VENT
AIR CHAMBER VENT
FRONT PANEL

Figure 3-10. Front panel detail. The front instrument panel contains the controls for regulating the flow rate of chilled water passing through the condenser, the rate of air entering the solution, and the rate of carbon dioxide entering the air chamber. It also houses the filters on the vent lines from the solution tank and the air chamber. The filters are located on the front to allow easy changing.

PLANT EXPOSURE CHAMBER PARTS LIST

Air System

PART: AIR GAUGE 0-15 PSI
SUPPLIER: AIR COMPONENTS SPECIALTY
PART NUMBER: 29846
PRICE: $5.00
NEEDED: 2
MANUFACTURER:

PART: 12 VOLT SOLENOID VALVE
SUPPLIER: BAY PNEUMATIC
PART NUMBER: EVO-13
PRICE: $16.00
NEEDED: 2
MANUFACTURER: CLIPPARD

PART: 2-WAY MINIATURE CONTROL VALVE
SUPPLIER: AIR COMPONENTS SPECIALTY
PART NUMBER: TV-2S
PRICE: $5.50
NEEDED: 2
MANUFACTURER: CLIPPARD

PART: 1/4"OD X 1/8"ID COPPER TUBING
SUPPLIER: LOCAL HARDWARE
PART NUMBER:
PRICE: $.43/FT
NEEDED: 4 FT
MANUFACTURER:

PART: SS QUICK CONNECT 1/8"OD TUBING
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-QM2-S-200 STEM
SS-QM2-B-200 BODY
PRICE: STEM $12.20
BODY $30.40
NEEDED: 2 EACH
MANUFACTURER: CRAWFORD

PART: HORIZONTAL BAR CLAMPS
SUPPLIER: MACKINNON TOOL & MACHINERY CO.
PART NUMBER: 52060
PRICE: $4.67
NEEDED: 8
MANUFACTURER: DE-STA-CO

PART: 1/4"OD X 1/8"ID POLYETHYLENE TUBING (BEVALINE 4)
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0404-037
PRICE: $23.53/100' ROLL
NEEDED: 25 FT
MANUFACTURER: RYAN HERCO

PART: 32" X 32" ALUMINUM CYLINDER WITH 2 BANDS
SUPPLIER: LOCAL SHEET METAL CO.
PART NUMBER:
PRICE: $200.00
NEEDED: 1
MANUFACTURER: LOCAL SHEET METAL CO.

PART: TEFLON FILM
SUPPLIER: LIVINGSTONE COATING CORP.
PART NUMBER: D-2366
PRICE: $.277/SQ FT
NEEDED: 36 SQ FT
MANUFACTURER: DUPONT

PART: PRESSURE SENSITIVE TEFLON OVERLAY
SUPPLIER: COLE-PARMER
PART NUMBER: K-6804-00
PRICE: $5.43/FT
NEEDED: 2 FT
MANUFACTURER: COLE-PARMER

PART: 12 VOLT ELECTRIC IMPELLER MOTOR
SUPPLIER: MICRONEL
PART NUMBER: C28776
PRICE: $49.75
NEEDED: 1
MANUFACTURER: MICRONEL

PART: SUBMINIATURE PRESSURE REGULATOR
SUPPLIER: AIRTRONICS
PART NUMBER: R-900-30
PRICE: $19.99
NEEDED: 2
MANUFACTURER: AIR LOGIC

PART: 3/4" MACHINE BOLTS, ROUND HEAD, 6/32
SUPPLIER: LOCAL HARDWARE
PART NUMBER:
PRICE: $2.25/100
NEEDED: 36
MANUFACTURER:

PART: SS CAPILLARY LINE .006"ID X 1/16"OD
SUPPLIER: SUPERIOR TUBE CO.
PART NUMBER:
PRICE: $3.6162/FT
NEEDED: 5 FT
MANUFACTURER: SUPERIOR TUBE CO.

PART: 1 CC SYRINGE
SUPPLIER: VWR
PART NUMBER: BD5625
PRICE:
NEEDED: 2
MANUFACTURER: BECTON-DICKINSON

PART: CLOSED CELL WEATHERSTRIPPING 1/2" X 3/8"
SUPPLIER: LOCAL LUMBER YARD
PART NUMBER:
PRICE: $1.19/10 FT
NEEDED: 12 FT
MANUFACTURER:

PART: 10/32 TO 1/8"ID HOSE BARB (BRASS)
SUPPLIER: BAY PNEUMATIC
PART NUMBER: 11752-1
PRICE: $.27
NEEDED: 12
MANUFACTURER: CLIPPARD

PART: 1/8"OD X 1/16"ID COPPER TUBING
SUPPLIER: LOCAL HARDWARE
PART NUMBER:
PRICE: $.27/FT
NEEDED: 6 FT
MANUFACTURER:

PART: QUICK DISCONNECTS
SUPPLIER: GEORGE WARDEN CO.
PART NUMBER: MCD 10-02 BODY
MC 22-02 INSERT
PRICE: BODY $4.67
INSERT $2.10
NEEDED: 2 EACH
MANUFACTURER: COLDER PRODUCTS CO.

PART: GASKETS FOR 10/32 FITTINGS
SUPPLIER: AIR COMPONENTS SPECIALTY
PART NUMBER: 11761-2
PRICE: $.35/PACKAGE
NEEDED: 10
MANUFACTURER: CLIPPARD

PART: 10/32 TO 1/8"OD TUBING CONNECTOR
SUPPLIER: AIR COMPONENTS SPECIALTY
PART NUMBER: 11923
PRICE: $.30
NEEDED: 10
MANUFACTURER: CLIPPARD

PART: POLYETHYLENE TUBING 5/8"OD X 1/2"ID
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0525-040
PRICE: $.166/FT
NEEDED: 8 FT
MANUFACTURER:

PART: 1/4"OD TO 1/16"OD REDUCER UNION
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-400-6-1
PRICE: $8.15
NEEDED: 1
MANUFACTURER: CRAWFORD

PART: 3/16"OD X 1/16"ID POLYETHYLENE TUBING (BEVALINE 4)
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0404-015
PRICE: $.21/FT
NEEDED: 2 FT
MANUFACTURER:

PART: 1/8"ID TUBE TO LUER LOCK SYRINGE FITTING
SUPPLIER: VALUE PLASTICS, INC.
PART NUMBER: SERIES FTL30
PRICE: $.10
NEEDED: 2
MANUFACTURER: VALUE PLASTICS, INC.

PART: MASS FLOW CONTROLLER
SUPPLIER: TYLAN CORP.
PART NUMBER: FC-260
PRICE: $675.00
NEEDED: 1
MANUFACTURER: TYLAN CORP.

PART: SINGLE EDGED ALUMINUM EXTRUSION
SUPPLIER: BARRETT, SQUIBB & ASSOC.
PART NUMBER: ALE-11L-12
PRICE: $17.30/FT.
NEEDED: 24 FT
MANUFACTURER: AMCO

PART: ALUMINUM CASTINGS CORNER ASSEMBLY
SUPPLIER: BARRETT, SQUIBB & ASSOC.
PART NUMBER: ALC-1L
PRICE: $2.20
NEEDED: 4
MANUFACTURER: AMCO

PART: LOCKING CLIP
SUPPLIER: BARRETT, SQUIBB & ASSOC.
PART NUMBER: ALC-SL
PRICE: $.09
NEEDED: 150
MANUFACTURER: AMCO

PART: SOLID ALUMINUM ROD
SUPPLIER: LOCAL MACHINE SHOP
PART NUMBER: 5/8" DIA
PRICE: $1.57/FT
NEEDED: 1 FT
MANUFACTURER:

PART: CART WHEELS
SUPPLIER: MCMASTER-CARR SUPPLY CO.
PART NUMBER: 2834T12
PRICE: $2.80
NEEDED: 4
MANUFACTURER:

PART: 3/4" A/C EXTERIOR PLYWOOD
SUPPLIER: LOCAL LUMBER YARD
PART NUMBER:
PRICE: $18.20/32 SQ FT
NEEDED: 9 SQ FT
MANUFACTURER:

PART: POLYETHYLENE SHEET 1/4"
SUPPLIER: LOCAL PLASTICS SUPPLIER
PART NUMBER:
PRICE: $71.50 /36 SQ FT
NEEDED: 9 SQ FT
MANUFACTURER:

PART: EPOXY PAINT
SUPPLIER: LOCAL PAINT STORE
PART NUMBER:
PRICE: $14.15/QUART
NEEDED: 4 QUARTS
MANUFACTURER:

PART: 16 GAUGE ALUMINUM SHEET
SUPPLIER: LOCAL SHEET METAL SHOP
PART NUMBER:
PRICE: $2.50/SQ FT
NEEDED: 4 SQ FT
MANUFACTURER:

PART: TEMP-R-TAPE
SUPPLIER: BOYD INC.
PART NUMBER: 3" WIDTH
PRICE: $114.30/36 YD ROLL
NEEDED: 8 FT
MANUFACTURER: CHR INDUSTRIES

PART: TEMP-R-TAPE (TEFLON TAPE)
SUPPLIER: BOYD INC.
PART NUMBER: 3/4" WIDTH
PRICE: $25.58/36 YD ROLL
NEEDED: 3 FT
MANUFACTURER: CHR INDUSTRIES

PART: ALUMINUM CASTINGS(CORNER & SIDE ASSEMBLY)
SUPPLIER: BARRETT, SQUIBB & ASSOC.
PART NUMBER: ALC-2L
PRICE: $3.10
NEEDED: 4
MANUFACTURER: AMCO

PART: ALUMINUM CASTINGS(TEE ASSEMBLY)
SUPPLIER: BARRETT, SQUIBB & ASSOC.
PART NUMBER: ALC-4L
PRICE: $3.15
NEEDED: 4
MANUFACTURER: AMCO

PART: DOUBLE EDGED ALUMINUM EXTRUSION
SUPPLIER: BARRETT, SQUIBB & ASSOC.
PART NUMBER: ALE-13L-12
PRICE: $23.00/FT
NEEDED: 7.5 FT
MANUFACTURER: AMCO

PART: EDGELESS ALUMINUM EXTRUSION
SUPPLIER: BARRETT, SQUIBB & ASSOC.
PART NUMBER: ALE-L-12
PRICE: $11.50/FT
NEEDED: 4 FT
MANUFACTURER: AMCO

Hydroponic System

PART: 2"ID POLYETHYLENE PIPE
SUPPLIER: MASKELL-ROBBINS, INC.
PART NUMBER:
PRICE: $1.17/FT
NEEDED: 14 FT
MANUFACTURER: PHILLIPS DRISCOPIPE, INC.

PART: BLACK POLYETHYLENE TUBING
SUPPLIER: CHAPIN WATERMATICS
PART NUMBER: BLACK #4
PRICE: $27.00/1000 FT
NEEDED: 15 FT
MANUFACTURER:

PART: 12 VOLT FAN
SUPPLIER: MICRONEL
PART NUMBER: V605L-12V
PRICE: $104.40
NEEDED: 1
MANUFACTURER: MICRONEL

PART: RT. ANGLE SHUT OFF VALVE
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-1GS4-A
PRICE: $34.50
NEEDED: 1
MANUFACTURER: WHITNEY

PART: RT.ANGLE REGULATING VALVE
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: B-1RF4-A
PRICE: $16.60
NEEDED: 1
MANUFACTURER: WHITNEY

PART: SOLID RUBBER STOPPERS
SUPPLIER: VWR
PART NUMBER: NO. 10.5
PRICE: $4.30/LB
NEEDED: 3
MANUFACTURER: VWR

PART: 6/32 SS MACHINE SCREW NUTS
SUPPLIER: LOCAL HARDWARE
PART NUMBER:
PRICE: $3.00/100
NEEDED: 12
MANUFACTURER:

PART: 3/8" NPT SS UNION BALL JOINT
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-6-UBJ
PRICE: $36.40
NEEDED: 1
MANUFACTURER: CAJON

PART: CLEAR VINYL TUBING 3/8"ID X 5/8"OD
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0500-111
PRICE: $49.50/100 FT ROLL
NEEDED: 1 FT
MANUFACTURER:

PART: 12 VOLT SS SOLENOID VALVE
SUPPLIER: KIP INC.
PART NUMBER: 14016
PRICE: $18.50
NEEDED: 3
MANUFACTURER: KIP INC.

PART: 1/2" NPT LIQUID LEVEL SWITCH
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 6905-100
PRICE: $17.25
NEEDED: 1
MANUFACTURER:

PART: CENTRIFUGAL PUMP
SUPPLIER: COLE-PARMER INSTRUMENT CO.
PART NUMBER: 101-605
PRICE: $270.00
NEEDED: 1
MANUFACTURER: MICROPUMP

PART: PRESSURE TRANSDUCER
SUPPLIER: MICROSWITCH (HONEYWELL)
PART NUMBER: 142PCOID
PRICE: $72.00
NEEDED: 2
MANUFACTURER: HONEYWELL

PART: 5/8"ID PIPE INSULATION
SUPPLIER: THERMAL SUPPLY
PART NUMBER:
PRICE: $.40/FT
NEEDED: 1 FT
MANUFACTURER:

PART: MINIPHONE PLUG TO MINIPHONE PLUG
SUPPLIER: LOCAL ELECTRONICS STORE
PART NUMBER:
PRICE: $1.49
NEEDED: 5
MANUFACTURER:

PART: HEAT SHRINK TUBING
SUPPLIER: LOCAL ELECTRONICS STORE
PART NUMBER: 278-1672
PRICE: $1.54/BOX
NEEDED: 1
MANUFACTURER: RADIO SHACK

PART: THERMISTORS
SUPPLIER: NEWARK ELECTRONICS
PART NUMBER: 30F1797 UNI-CURVE
PRICE: $4.72
NEEDED: 5
MANUFACTURER:

PART: 3/16"OD,.016" WALL,304 GRADE SS TUBING
SUPPLIER: TUBESALES
PART NUMBER:
PRICE: $2.50/FT
NEEDED: 2 FT
MANUFACTURER: 1

PART: 3/8" NPT SS MALE TEE
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-6-MT
PRICE: $13.10
NEEDED: 1
MANUFACTURER: CAJON

PART: HOSE CLAMPS 13/16"
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0925-016
PRICE: $.29
NEEDED: 9
MANUFACTURER:

PART: SS QUICK CONNECT 1/8"OD TUBING
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-QC4-S-4HC STEM
SS-QC4-B1-400 BODY
PRICE: STEM $11.20
BODY $35.60
NEEDED: 1 EACH
MANUFACTURER: CRAWFORD

PART: 1/4" NPT SS HEX NIPPLE
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-4-HN
PRICE: $3.40
NEEDED: 1
MANUFACTURER: CAJON

PART: 3/8" NPT SS HEX NIPPLE
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-6-HN
PRICE: $3.80
NEEDED: 1
MANUFACTURER:

PART: 1/8" NPT TO 1/4"OD TUBE CONNECTOR
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-400-1-2
PRICE: $4.45
NEEDED: 10
MANUFACTURER: CRAWFORD

PART: 1/8" NPT TO 5/32"ID HOSE BARB
SUPPLIER: VALUE PLASTICS INC.
PART NUMBER: 1804-2
PRICE: $.22
NEEDED: 25
MANUFACTURER: VALUE PLASTICS

PART: 3/8" NPT TO 5/9"ID HOSE BARB ELBOW
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0710-173
PRICE: $.35
NEEDED: 13
MANUFACTURER:

PART: 1/8" NPT TO 3/16"ID THREADED HOSE BARB ELBOW
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0710-150
PRICE: $.32
NEEDED: 2
MANUFACTURER:

PART: PH PROBE 3/4" NPT PVC NIPPLE
SUPPLIER: ANALYTICAL MEASUREMENTS INC.
PART NUMBER:
PRICE: $35.00
NEEDED: 1
MANUFACTURER: ANALYTICAL MEASUREMENTS

PART: CONDUCTIVITY PROBE
SUPPLIER: VWR
PART NUMBER: 414
PRICE: $114.00
NEEDED: 1
MANUFACTURER: BECKMAN

PART: FEMALE BRANCH TEE 1/8" NPT TO 1/8"OD TUBE
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: B-200-3TTF
PRICE: $6.75
NEEDED: 2
MANUFACTURER: CRAWFORD FITTING COMPANY

PART: 3/8" NPT TO 3/8"ID HOSE BARB
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: B-6-HC-1-6
PRICE: $2.30
NEEDED: 2
MANUFACTURER: CAJON COMPANY

PART: 1/4" NPT TO 1/2"ID HOSE BARB ELBOW
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0710-165
PRICE: $.35
NEEDED: 20
MANUFACTURER:

PART: HOSE CLAMP 9/16"
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0950-008
PRICE: $.85
NEEDED: 2
MANUFACTURER:

PART: 3/8" NPT TO 1/4"ID HOSE BARB ELBOW
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0710-168
PRICE: $.35
NEEDED: 3
MANUFACTURER:

PART: 3/8" NPT TO 1/2"ID HOSE BARB
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0700-173
PRICE: $.20
NEEDED:
MANUFACTURER: 3

PART: 1/4"ID X 3/8"OD POLYETHYLENE TUBING
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0404-070
PRICE: $43.11/100 FT
NEEDED: 3 FT
MANUFACTURER:

PART: 5/8"ID X 13/16"OD POLYETHYLENE TUBING
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0404-164
PRICE: $149.49/100FT.
NEEDED: 20 FT.
MANUFACTURER:

PART: 3"ID POLYETHYLENE PIPE
SUPPLIER: MASKELL-ROBBINS INC.
PART NUMBER:
PRICE: $2.57/FT
NEEDED: 1/2 FT
MANUFACTURER: PHILLIPS DRISCOPIPE,INC

PART: 5"ID POLYETHYLENE PIPE
SUPPLIER: MASKELL-ROBBINS INC.
PART NUMBER:
PRICE: $6.54/FT
NEEDED: 1/2 FT
MANUFACTURER: PHILLIPS DRISCOPIPE INC.

PART: 8"ID POLYETHYLENE PIPE
SUPPLIER: MASKELL-ROBBINS INC.
PART NUMBER:
PRICE: $8.50/FT
NEEDED: 1 FT
MANUFACTURER: PHILLIPS DRISCOPIPE, INC.

PART: VACUTAINER SEPTUM
SUPPLIER: VWR
PART NUMBER: BD 6493
PRICE: $94.00/1000
NEEDED: 1
MANUFACTURER: BECTON-DICKINSON

PART: QUICK CONNECT 3/8"OD TUBE
DOUBLE END SHUT OFF
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: STEM:B-QC6-D-600
BODY:B-QC6-B1-600
PRICE: STEM $9.80, BODY $11.80
NEEDED: 2 EACH
MANUFACTURER: CRAWFORD FITTING COMPANY

PART: 5/8"OD X 5/8"OD BARBED ELBOW
SUPPLIER: RYAN HERCO PRODUCTS CORP.
PART NUMBER: 0710-028
PRICE: $.36
NEEDED: 8
MANUFACTURER:

PART: 3/4"ID PIPE INSULATION
SUPPLIER: THERMAL SUPPLY
PART NUMBER:
PRICE: $.42 FT
NEEDED: 15 FT
MANUFACTURER:

PART: SS PAN
SUPPLIER: LOCAL RESTAURANT SUPPLY
PART NUMBER:
PRICE: $8.00
NEEDED: 1
MANUFACTURER:

PART: QUICK DISCONNECTS
SUPPLIER: GEORGE WARDEN CO.
PART NUMBER: MCD 10-02 BODY
MC 22-02 STEM
PRICE: BODY $4.67
STEM $2.10
NEEDED: 1 EACH
MANUFACTURER: COLDER PRODUCTS

PART: 1/8" FEMALE PIPE TO 1/4"OD TUBE
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-400-7-2
PRICE: $6.50
NEEDED: 1
MANUFACTURER: CRAWFORD

PART: PLUG VALVE
SUPPLIER: PORTLAND VALVE AND FITTING
PART NUMBER: SS-4P4T4
PRICE: $29.90
NEEDED: 1
MANUFACTURER: NUPRO COMPANY

MANUFACTURERS AND SUPPLIERS

CO. NAME: AIR COMPONENTS SPECIALTY
ST. ADDRESS: 433 SE 85TH
CITY, STATE: PORTLAND, OR
ZIP: 97216
PHONE: (503) 257-9848

CO. NAME: AIR LOGIC
ST. ADDRESS: 5102 DONGLAS AVE.
CITY, STATE: RACINE, WI
ZIP: 53402
PHONE: (414) 639-9035

CO. NAME: AIRTRONICS
ST. ADDRESS: 2020 124 AVE. NE
BUILDING 101C
CITY, STATE: BELLEVUE, WA
ZIP: 98005
PHONE: (206) 881-6443

CO. NAME: AMED ENGINEERING CO.
ST. ADDRESS: 3801 N. ROSE STREET
CITY, STATE: SCAILLER PARK, IL
ZIP: 60176
PHONE: (312) 617-6670

CO. NAME: ANALYTICAL MEASUREMENTS INC.
ST. ADDRESS: 31 WILLOW ST.
CITY, STATE: CHATHAM, NJ
ZIP: 07928
PHONE: (201) 273-7500

CO. NAME: BARRETT, SQUIBB & ASSOC.
ST. ADDRESS: 14273 NW SCIENCE PARK DR.
SUITE 500
CITY, STATE: PORTLAND, OR
ZIP: 97229
PHONE: (503) 644-2550

CO. NAME: BAY PNEUMATIC
ST. ADDRESS: 440 CONVENTION WAY
CITY, STATE: REDWOOD CITY, CA
ZIP: 94063
PHONE: (415) 365-1444

CO. NAME: BECKMAN
ST. ADDRESS: 15511 NE 90TH ST.
CITY, STATE: REDMOND, WA
ZIP: 98052
PHONE: (800) 426-8003

CO. NAME: BECTON-DICKINSON LABWARE
ST. ADDRESS: P.O. BOX 310
CITY, STATE: RUTHERFORD, NJ
ZIP: 07070
PHONE: (201) 460-2214

CO. NAME: BOYD INC.
ST. ADDRESS: 6136 NE 87TH AVE.
CITY, STATE: PORTLAND, OR
ZIP: 97220
PHONE: (503) 225-5405

CO. NAME: CAJON COMPANY
ST. ADDRESS: 9760 SHEPARD RD.
CITY, STATE: MACEDONIA, OH
ZIP: 44050
PHONE: (216) 467-0200

CO. NAME: CHAPIN WATERMATICS
ST. ADDRESS: 368 N. COLORADO AVE.
CITY, STATE: WATERTOWN, NY
ZIP: 13601
PHONE: (315) 782-117

CO. NAME: CHR INDUSTRIES INC.
(AN ARMCO CO.)
ST. ADDRESS: 407 EAST ST.
CITY, STATE: NEW HAVEN, CT
ZIP: 06509
PHONE: (203) 777-3631

CO. NAME: CLIPPARD INSTRUMENT LABORATORY INC.
ST. ADDRESS: 7390 COLERAIN ROAD
CITY, STATE: CINCINNATI, OH
ZIP: 45239
PHONE: (513) 521-4261

CO. NAME: COLDER PRODUCTS CO.
ST. ADDRESS: 2459 UNIVERSITY AVE.
CITY, STATE: ST.PAUL, MN
ZIP: 55114
PHONE: (612) 645-0091

CO. NAME: COLE-PARMER INSTRUMENT CO.
ST. ADDRESS: 7425 NORTH OAK PARK AVE.
CITY, STATE: CHICAGO, IL
ZIP: 60648
PHONE: (800) 323-4340

CO. NAME: CRAWFORD FITTING COMPANY
ST. ADDRESS: 29500 SOLON ROAD
CITY, STATE: SOLON, OH
ZIP: 44139
PHONE: (216) 248-4600

CO. NAME: DE-STA-CO. DIVISION
DOVER CORPORATION
ST. ADDRESS: 350 MIDLAND AVE.
CITY, STATE: DETROIT, MI
ZIP: 48203
PHONE: (318) 868-5080

CO. NAME: GEORGE WARDEN CO.
ST. ADDRESS: 9020 NE VANCOUVER WAY
CITY, STATE: PORTLAND, OR
ZIP: 97211
PHONE: (800) 452-6331/(503) 2889-8891

CO. NAME: H-O PRODUCTS CORPORATION
ST. ADDRESS: 10 BRIDGE STREET
P.O. BOX 526
CITY, STATE: WINSTED, CT
ZIP: 06098
PHONE: (203) 379-9875

CO. NAME: KIP INC.
ST. ADDRESS: 13 TOWN LINE RD.
P.O. BOX 6162
CITY, STATE: WOLCOTT, CT
ZIP: 06710
PHONE: (203) 677-0272

CO. NAME: LIVINGSTONE COATING CORP.
ST. ADDRESS: 2400 RHYNE RD.
P.O.BOX 668282
CITY, STATE: CHARLOTTE, NC
ZIP: 28266
PHONE: (704) 392-2323

CO. NAME: MACKINNON TOOL AND
MACHINERY CO.
ST. ADDRESS: P.O. BOX 14647
1001 SE SANDY RD.
CITY, STATE: PORTLAND, OR
ZIP: 97241
PHONE: (800) 452-8408/(503 235-8555

CO. NAME: MASKELL-ROBBINS,INC.
ST. ADDRESS: 3733 CHERRY AVE. NE
CITY, STATE: SALEM, OR
ZIP: 97303
PHONE: (503) 393-7921

CO. NAME: MCMASTER-CARR SUPPLY CO.
ST. ADDRESS: P.O. BOX 54960
CITY, STATE: LOS ANGELES, CA
ZIP: 90054
PHONE: (213) 945-1311

CO. NAME: MICRONEL
ST. ADDRESS: 4716 S. MISSION RD.
CITY, STATE: FALLBROOK, CA
ZIP: 92028
PHONE: (714) 723-9505

CO. NAME: MICROPUMP
ST. ADDRESS: P.O. BOX 4001
CITY, STATE: CONCORD, CA
ZIP: 94527
PHONE: (415) 687-0101

CO. NAME: MICROSWITCH (HONEYWELL)
ST. ADDRESS: 6420 SW MACADAM
CITY, STATE: PORTLAND, OR
ZIP: 97201
PHONE: (503) 245-0721

CO. NAME: NEWARK ELECTRONICS
ST. ADDRESS: 11300 NE HALSEY
CITY, STATE: PORTLAND, OR
ZIP: 97220
PHONE: (503) 257-0741

CO. NAME: NUPRO COMPANY
ST. ADDRESS: 4800 EAST 345TH STREET
CITY, STATE: WILLOUGHBY, OH
ZIP: 44094
PHONE: (216) 951-7100

CO. NAME: PACIFIC METAL CO.
ST. ADDRESS: 535 CONGER
CITY, STATE: EUGENE, OR
ZIP: 97402
PHONE: (503) 485-1876

CO. NAME: PHILLIPS DRISCOPIPE CO.
ST. ADDRESS: P.O. BOX 508
CITY, STATE: WATSONVILLE, CA
ZIP: 95076
PHONE: (408) 722-8166

CO. NAME: PORTLAND VALVE AND FITTING
ST. ADDRESS: 703 N. RUSSELL ST.
CITY, STATE: PORTLAND, OR
ZIP: 97227
PHONE: (503) 288-6901

CO. NAME: RYAN HERCO PRODUCTS CORP.
ST. ADDRESS: P.O. BOX 588
CITY, STATE: BURBANK, CA
ZIP: 91503
PHONE: (800) 423-2589

CO. NAME: SUPERIOR TUBE CO.
ST. ADDRESS: BOX 191
CITY, STATE: NORRISTOWN, PA
ZIP: 19404
PHONE: (214) 489-5200

CO. NAME: TECHNICAL CONTROLS INC.
ST. ADDRESS: 3247 NW 29TH AVE
CITY, STATE: PORTLAND, OR
ZIP: 97210
PHONE: (800) 452-0953

CO. NAME: THERMAL SUPPLY
ST. ADDRESS: 1407 WEST 7TH AVE.
CITY, STATE: EUGENE, OR
ZIP: 97402
PHONE: (800) 452-6077/(503) 687-8311

CO. NAME: TUBESALES
ST. ADDRESS: 5319 SW WESTGATE DR.
SUITE 105
CITY, STATE: PORTLAND, OR
ZIP: 97221
PHONE: (800) 452-1644/(503) 297-3366

CO. NAME: TYLAN CORP.
ST. ADDRESS: 23301 SO.WILMINGTON AVE.
CITY, STATE: CARSON, CA
ZIP: 90745
PHONE: (213) 518-3610

CO. NAME: VALUE PLASTICS INC.
ST. ADDRESS: 626 WEST 60TH
CITY, STATE: LOVELAND, CO
ZIP: 90537
PHONE: (303) 669-4351

CO. NAME: VWR
ST. ADDRESS: P.O. BOX 14070
CITY, STATE: PORTLAND, OR
ZIP: 97214
PHONE: (503) 234-9272

CO. NAME: WHITEY CO.
ST. ADDRESS: 318 BISHOP RD.
CITY, STATE: HIGHLAND HEIGHTS, OH
ZIP: 44143
PHONE: (216) 473-1050

APPENDIX 4

ELECTRONIC DIAGRAMS FOR COMPONENTS OF THE PLANT EXPOSURE LABORATORY

by

John Wilson

Bendix Field Engineering Corp.
Division of Allied Bendix Aerospace
Corvallis Environmental Research Laboratory
200 S.W. 35th Street
Corvallis, Oregon 97333

CORVALLIS ENVIRONMENTAL RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CORVALLIS, OREGON 97333

CONTENTS

Page
Figures 4-iv
Introduction 4-1
Plant Exposure Chamber Electronics 4-2
Plant Exposure Laboratory Electronics 4-20
Literature Cited 4-26

INTRODUCTION

The electronic schematics presented in this appendix support the plant exposure laboratory and exposure chambers developed by scientists at EPA's Corvallis research laboratory. The electronics allow a small computer (Apple //e) to measure environmental parameters in up to 18 experimental chambers on a regular schedule. This requires switching circuits which sequentially select the designated chambers and interrogate their sensors. Some sensors share electronic circuits (e.g. temperature and electrical conductivity) and others depend on individual circuits mounted on each chamber (e.g. pH and light). Another switching system controls gas sampling by operating various valves and a pump to create a loop in which air from the designated chamber is pumped through a two-channel (CO_2 and H_2O) infrared gas analyzer (IRGA).

Control of various parameters on each chamber is facilitated by the circuits presented. Some may be preset (e.g. fan and impellar speeds) and others are computer operated (e.g. CO_2 injection and irrigation).

Descriptions of chamber functions and computer operations which use these circuits are contained in the main text and in Appendices 1 and 2.

Circuits have been arranged onto different printed circuit boards which plug into standard printed circuit board edge connectors in the electronic module of each chamber or into the general electronic support panel in the lab.

The circuits shown are the author's design unless otherwise noted.

PLANT EXPOSURE CHAMBER ELECTRONICS

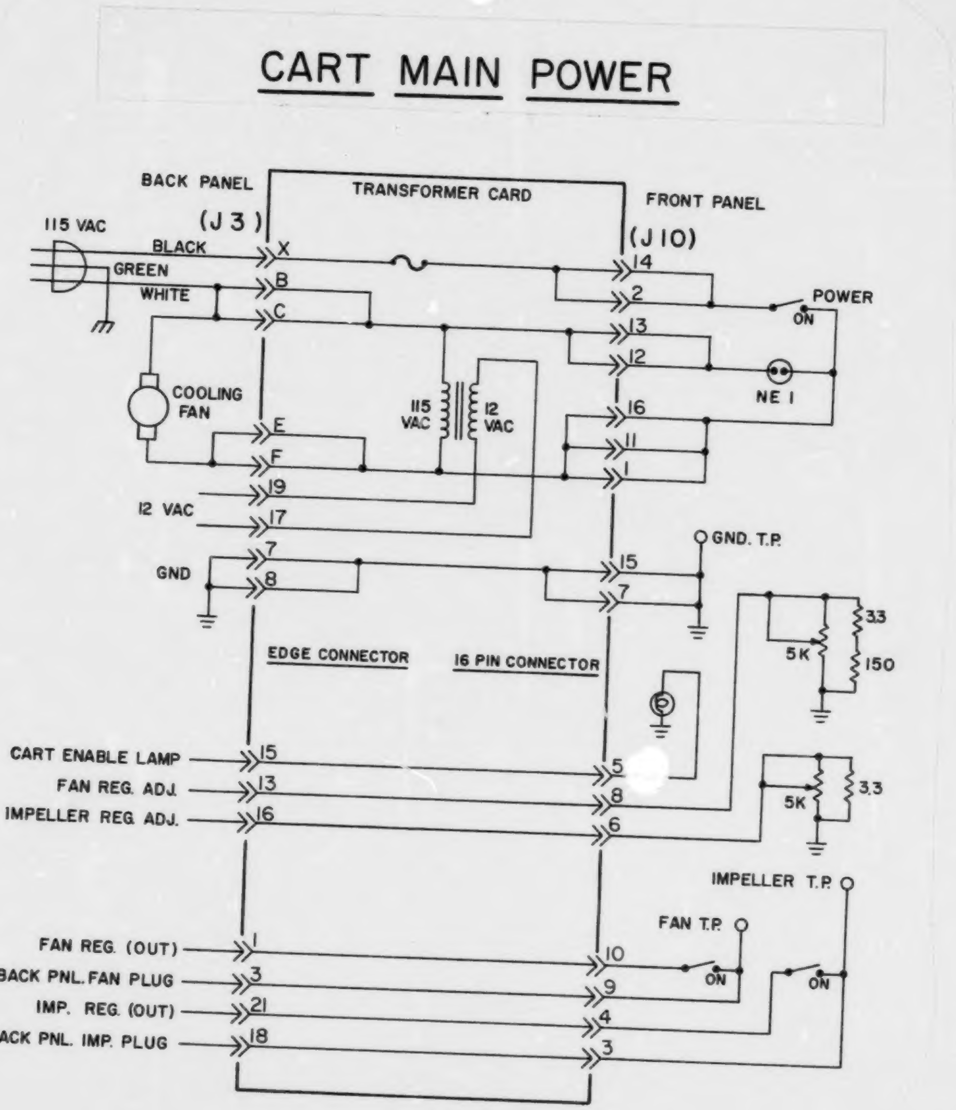

CART MAIN POWER
BACK PANEL
TRANSFORMER CARD
FRONT PANEL
115 VAC
(J 3)
(J 10)
BLACK
GREEN
WHITE
X
B
C
14
2
POWER
ON
13
12
NE 1
COOLING FAN
115 VAC
12 VAC
E
F
16
11
1
19
12 VAC
17
GND
7
8
GND. T.P.
15
7
33
5K
150
EDGE CONNECTOR
16 PIN CONNECTOR
CART ENABLE LAMP
FAN REG. ADJ.
IMPELLER REG. ADJ.
15
13
16
5
8
6
5K
3.3
IMPELLER T.P.
FAN T.P.
FAN REG. (OUT)
BACK PNL. FAN PLUG
IMP. REG. (OUT)
BACK PNL. IMP. PLUG
1
3
21
18
10
9
4
3
ON
ON

Figure 4-1. Cart main power card. Electricity (115 VAC) is connected to the transformer which supplies 12 VAC to the power supply card (Fig. 4-2), the mass flow controller code driver card (Fig. 4-14), and the solenoid driver card (Fig. 4-5). It also provides 115 VAC to the ON/OFF switch and the power light on the front panel of the electronics module.

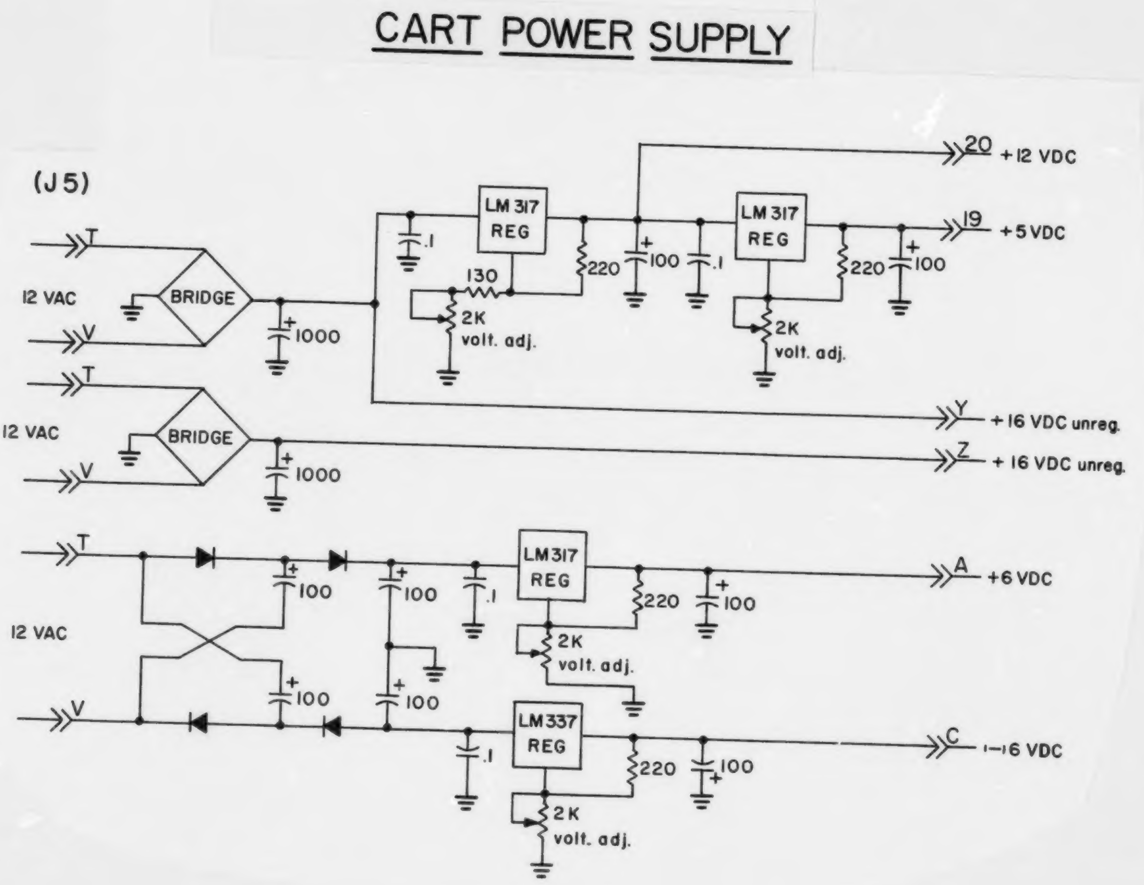

CART POWER SUPPLY
(J5)
T
12 VAC
V
BRIDGE
1000
T
12 VAC
V
BRIDGE
1000
LM 317 REG
.1
130
2K volt. adj.
220
100
.1
LM 317 REG
2K volt. adj.
220
100
20 +12 VDC
19 +5 VDC
Y +16 VDC unreg.
Z +16 VDC unreg.
T
12 VAC
V
100
100
100
100
.1
LM317 REG
2K volt. adj.
220
100
A +6 VDC
.1
LM337 REG
220
100
2K volt. adj.
C -16 VDC

Figure 4-2. Cart power supply card. 12 VAC current is converted to +6 VDC, -6 VDC, +12 VDC, and +5 VDC for various others cards in the electronic module. This card also supplies unregulated 16 VDC to the fan and impeller regulator card (Fig. 4-3).

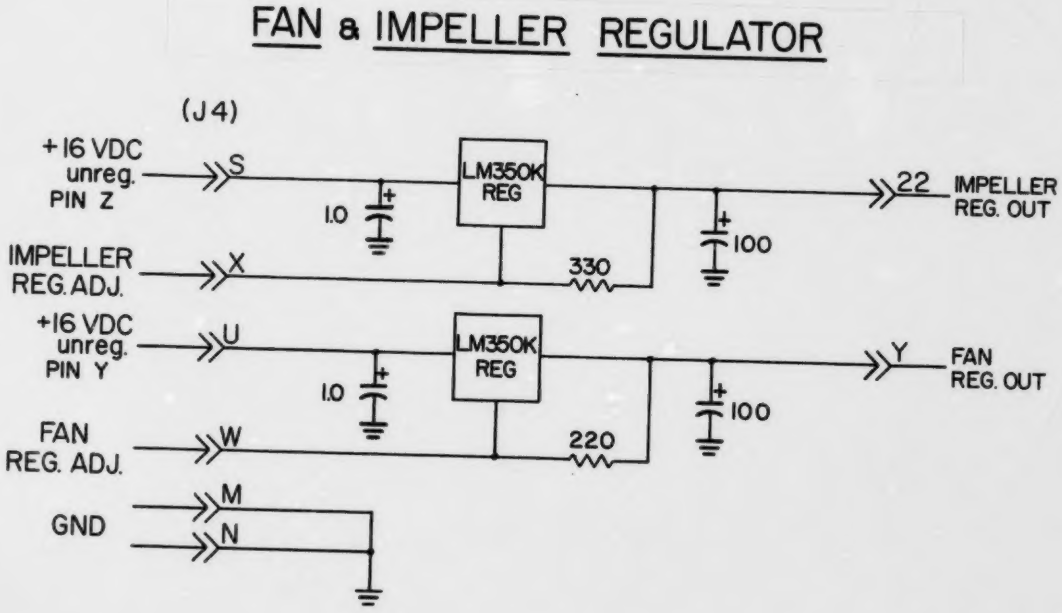

FAN & IMPELLER REGULATOR
(J4)
+16 VDC unreg. PIN Z
S
1.0
LM350K REG
22
IMPELLER REG. OUT
100
IMPELLER REG. ADJ.
X
330
+16 VDC unreg. PIN Y
U
1.0
LM350K REG
Y
FAN REG. OUT
100
FAN REG. ADJ.
W
220
GND
M
N

Figure 4-3. Fan and impeller regulator card. Regulated power up to 12 VDC is supplied to the fan and up to 9 VDC to the impeller. Impeller motors, although listed as 12 VDC, usually burn out if operated above 9 volts. Adjustments are located on the front panel of the electronics module.

CART ENABLE RELAYS

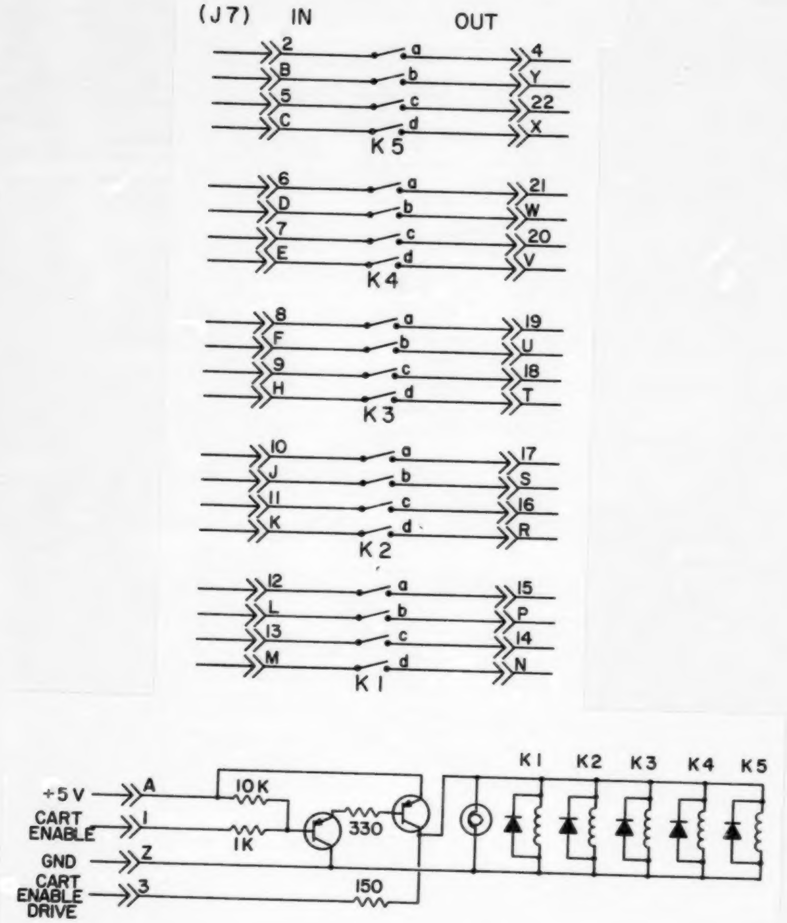

(J7) IN
OUT
2
B
5
C
a
b
c
d
4
Y
22
X
K5
6
D
7
E
21
W
20
V
K4
8
F
9
H
19
U
18
T
K3
10
J
11
K
17
S
16
R
K2
12
L
13
M
15
P
14
N
K1
K1 K2 K3 K4 K5
+5V
A
CART ENABLE
1
GND
Z
CART ENABLE DRIVE
3
10K
1K
330
150

Figure 4-4. Cart enable relay card. Eighteen signals come from the cart enable select circuitry. One signal goes to each of the 18 cart locations in the room. This makes each location unique and separate from all others. When a cart position is selected the cart enable signal for that cart goes from 0.0 VDC to 5.0 VDC. This activates the relays on the cart enable card in that electronic module only. The closed contacts apply the analog signals from that cart to the A to D bus. The cart enable light on the front of the electronics module indicates the cart is enabled. The computer goes through the A to D portion of the program and reads the voltages. The voltages are converted to the appropriate units and stored on disk. They are also displayed on the monitor. When finished the computer disables the cart by selecting the next cart in the sequence or "63" which is a nonexisting position.

Solenoid Driver Card

Various environmental conditions are controlled using these circuits. After finding a parameter out of tolerance, the computer activates the cart enable relays on the desired chamber and then sends a signal via the D109 to control the action of various valves. Since only one chamber can be enabled at any time, the correct chamber receives the command via this solenoid driver card to open or close the desired valve or make other adjustments.

The following functions may currently be managed by this system:

Drain the transpired water collector.
Add nutrient solution to the hydroponic tank or irrigant to soils.
Add acid or base to control solution pH.
Control CO_2 injection rate into chamber.
Open/close CO_2 delivery valve.
Open/close O_2 delivery valve.
Activate electrical conductivity measurement circuit.

Future:

Automatically sample solutions from hydroponic, leachate and transpiration collectors.

Two 12 volt regulators are used in the power supply on this card, dividing the solenoid loads with the two regulators. It was originally intended that many of the functions or solenoids would be operated at the same time thereby causing loading problems with the 12 volts. A serial approach is now used, so one of the regulators could be eliminated and its load tied to the other.

CART SOLENOID DRIVER

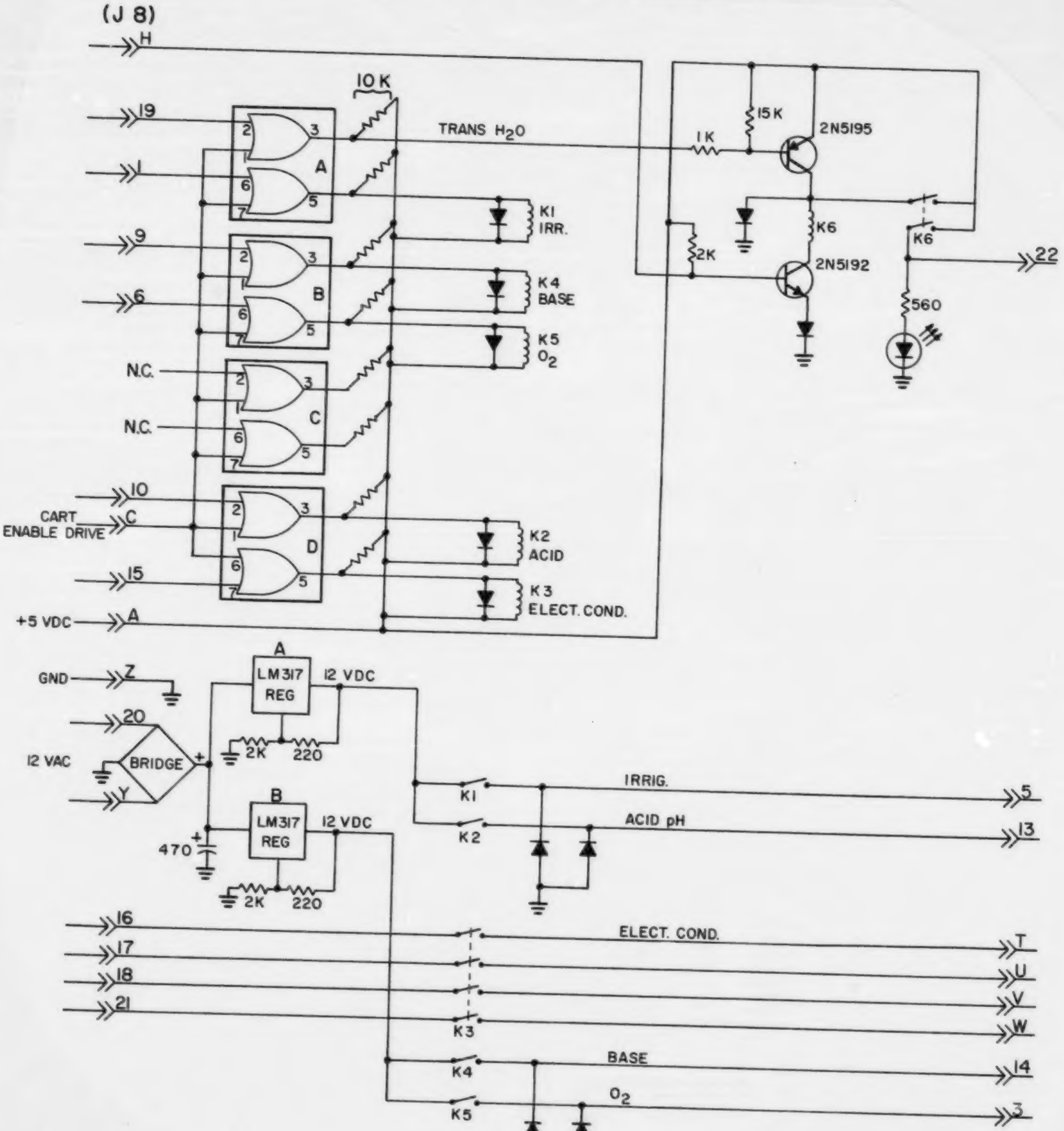

(J 8)
H
19
I
9
6
N.C.
N.C.
10
CART ENABLE DRIVE
C
15
+5 VDC
A
10 K
2
3
1
6
5
7
A
B
C
D
TRANS H_2O
K1 IRR.
K4 BASE
K5 O_2
K2 ACID
K3 ELECT. COND.
1 K
15 K
2N5195
K6
2K
2N5192
K6
22
560
GND
Z
20
12 VAC
BRIDGE
Y
+
LM317 REG
12 VDC
2K
220
470
K1
K2
IRRIG.
ACID pH
5
13
16
17
18
21
ELECT. COND.
T
U
V
W
K3
K4
K5
BASE
O_2
14
3

Figure 4-5. Cart solenoid driver card.

pH/Light Card

The pH/light card is composed of two circuits: The light amplifier circuit and the pH circuit.

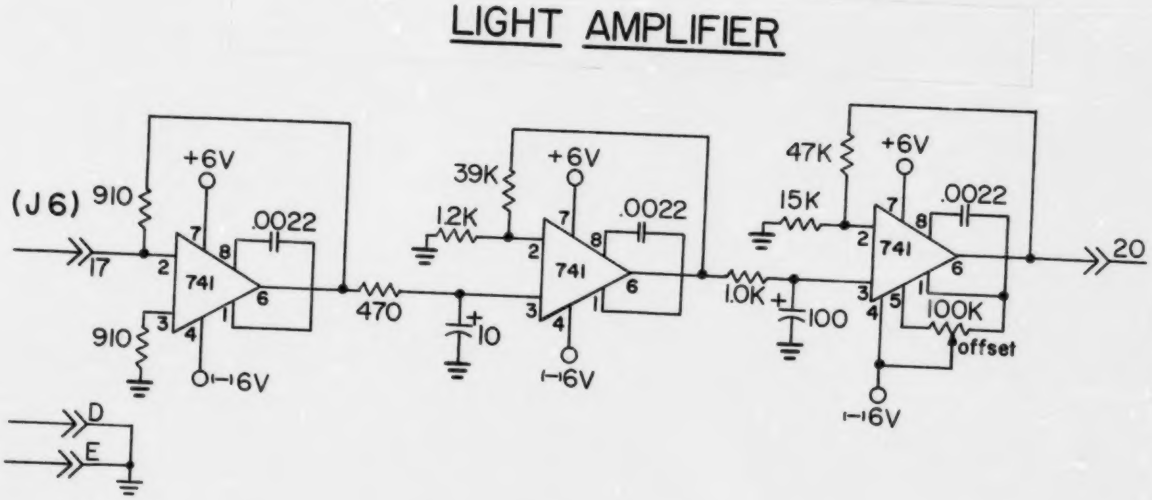

LIGHT AMPLIFIER
(J6)
910
17
2
3
910
741
+6V
7
8
.0022
6
4
-6V
470
+
10
39K
12K
+6V
7
2
3
741
8
.0022
6
1
4
-6V
1.0K
+
100
47K
+6V
15K
7
2
3
741
8
.0022
6
1
5
4
100K
offset
-6V
20
D
E

Figure 4-6. Light amplifier circuit. The circuit consists of a two-stage amplifier and a transconductance converter. The current to voltage converter is from Dobkin (1973). The output goes to channel 6 on the A to D converter.

Adjustment: With the light sensor covered adjust the output at pin 6 of the final amplifier stage to read 0.0 VDC. Remove the cover and place the sensor under a light source. Check the output at pin 6 again; it should be > 1.5 VDC.

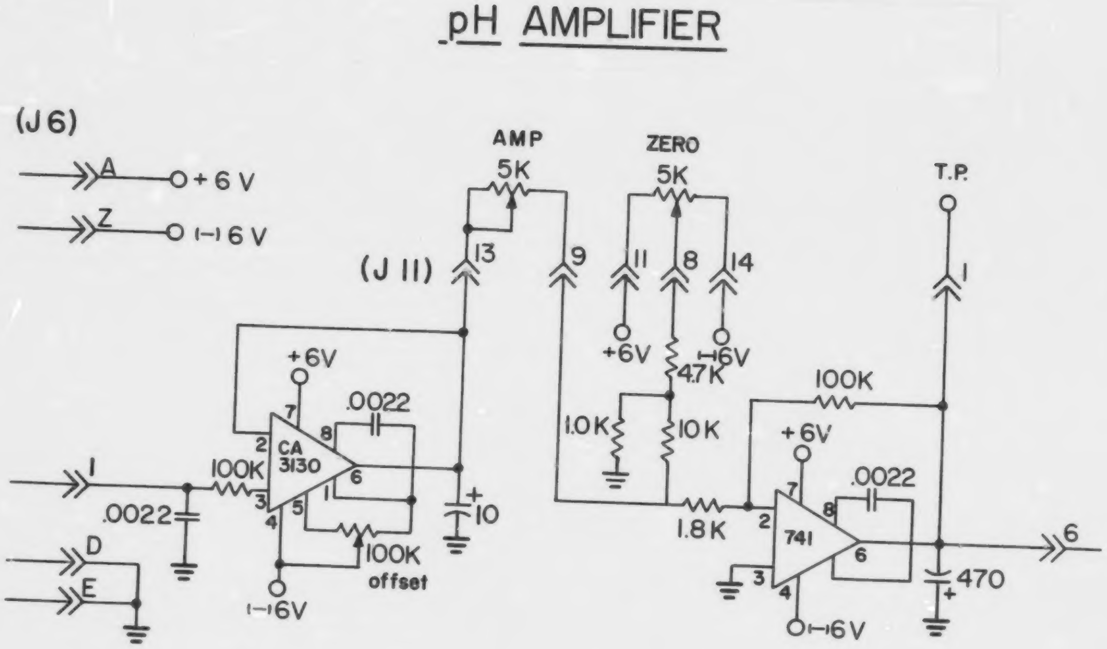

pH AMPLIFIER
(J6)
A
+6 V
Z
-6 V
AMP
5K
ZERO
5K
T.P.
(J 11)
13
9
11
8
14
1
+6V
-6V
4.7K
1.0K
10K
100K
+6V
.0022
2
7
8
CA
3130
6
3
4
5
1
100K
.0022
100K
offset
-6V
D
E
+
10
+6V
.0022
1.8K
2
741
3
4
6
8
7
-6V
470
+
6

Figure 4-7. pH circuit. The pH circuit is from Lancaster (1982). Its output goes to channel 5 on the A to D converter.

The pH probe is mounted on the cart in the probe pot. Its coaxial cable is connected to the back of the electronic module by a BNC connector. The BNC connector is connected internally to this printed circuit board by coaxial cable. The adjustment pots and output are on the face of the module for easy adjustment. Locking potentiometers are used to prevent accidental movement. A BNC connector on the probe is used so the probes can be easily removed.

Adjustment: With the probe in a pH 7 buffer, set the zero for 0.0 VDC. Put the probe in a pH 4 buffer and set the amplifier for -4.0 VDC. Check the probe in pH 7 buffer to see if it holds (sometimes adjustments are interacting).

THERMISTOR INTERFACE

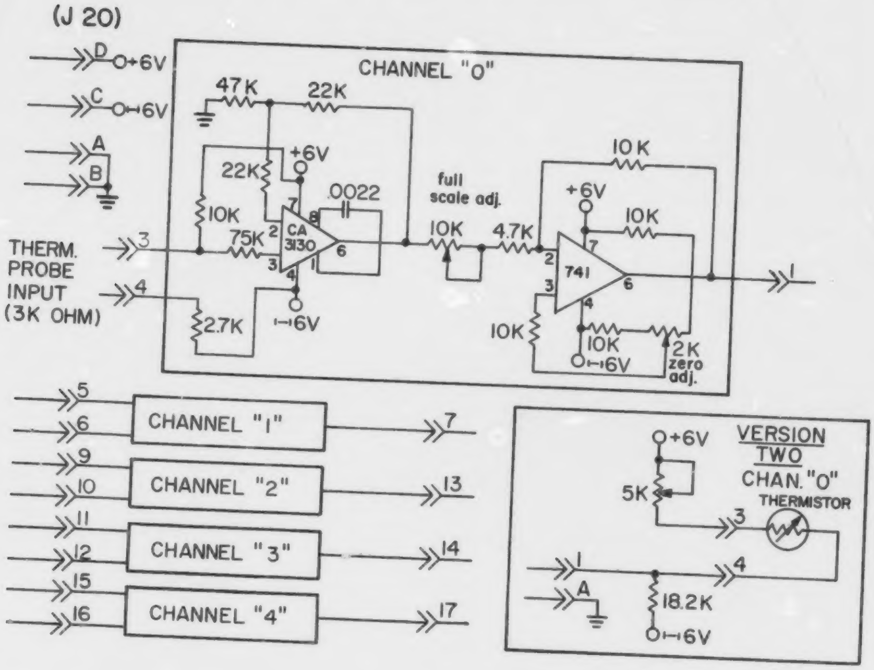

(J 20)
D
+6V
C
-6V
A
B
CHANNEL "O"
47K
22K
22K
+6V
7
.0022
8
10K
75K
2
CA 3130
6
3
1
4
-6V
THERM. PROBE INPUT (3K OHM)
3
4
2.7K
full scale adj.
10K
4.7K
10K
+6V
10K
7
2
741
3
6
4
10K
10K
-6V
2K zero adj.
1
5
6
CHANNEL "1"
7
9
10
CHANNEL "2"
13
11
12
CHANNEL "3"
14
15
16
CHANNEL "4"
17
+6V
VERSION TWO
CHAN. "O"
5K
THERMISTOR
3
1
4
A
18.2K
-6V

Figure 4-8. Thermistor interface card. Each cart can measure the output from up to five thermistors. The two leads for each thermistor come through the cart enable relays to the inputs (0-4) of the thermistor interface card located in the central electronics cabinet. The outputs of the thermistor interface card channels 0-4 go to channels 0-4 of the AI-13 (A to D converter) which is connected to the computer.

The thermistor completes the circuit in the temperature amplifier. The thermistor leads are isolated from the case of the electronic box by mounting them on a vector printed circuit board in the case and enlarging the case holes around the mini phone plugs so they do not touch the case. Thermistor assembly is described in Appendix 5.

Version two is an alternate means of construction. This method is useful for limiting amplifier drift.

Adjustment --

1. Put the probe into an ice slurry and adjust the zero potentiometer to 0.0 VDC on pin 6 of the 741 operational amplifier for that channel.
2. Put probe in a 35°C water bath and set amplification potentiometer for 3.5 VDC at pin 6 of 741.
3. Repeat steps 1 and 2 until the circuit output is 0.0 VDC and 3.5 VDC.
4. Repeat the adjustment procedure for all 5 channels.

Appendix 4

Wiring List for Backplane
(For Locations see Figure 4-9)

Backplane From	To	Signal Name
Line Cord		
Black	J3-X	115 VAC (hot)
White	J3-B, J3-C, Fan (muffin)	115 VAC (common)
Green	Case	Case ground
Fan (muffin)	J3-#, J3-F	115 VAC (switched, hot)
J3-1	J4-Y	Fan regulator
J3-3	Fan plug (back panel)	Fan voltage
J3-7, J3-8	J4-M, N, J5-K, L, M, N, P, J6-D, E, J7-Z, J8-Z, J1-31, 32, 33, 34, 35, 36, 37, J2-13, 14, 15, 16, 17, 18, 19, 20, J2-21, 22, 23, 24, 25, pH Connector, Light Connector, Fan Connector, Impeller Connector (back panel)	Ground
J3-13	J4-W	Fan reg. adjust
J3-15	J7-3	Cart enable led
J3-16	J4-X	Impeller reg. adjust
J3-17	J5-T, J8-20	12 VAC
J3-18	Impeller Plug (back panel)	Impeller voltage
J3-21	J4-22	Impeller regulator
J4-S	J5-Z	Rect. out (imp)
J4-U	J5-Y	Rect. out (fan)
J5-19	J7-A, J8-A	5 VDC
J5-20	J2-1	12 VDC
J5-A	J6-A	+6 VDC
J5-C	J6-Z	-6 VDC
J6-1	pH connector (back panel)	pH input
J6-6	J7-8	pH output
J6-17	Light connector (back panel)	Light input
J6-20	J7-F	Light output
J7-1	J1-40, J8-C	Cart enable
J7-5	J1-38	Reset CO_2
J7-6	J8-1	Irrigate output
J7-7	J2-4	T2
J7-9	Thermistor #4 connector (back panel)	
J7-H	Thermistor #4 connector (back panel)	
J7-10	Thermistor #2 connector (back panel)	

Backplane

From	To	Signal Name	
J7-J	Thermistor #2 connector (back panel)		
J7-11	Thermistor #3 connector (back panel)		
J7-K	Thermistor #3 connector (back panel)		
J7-12	Thermistor #0 connector (back panel)		
J7-L	Thermistor #0 connector (back panel)		
J7-13	Thermistor #1 connector (back panel)		
J7-M	Thermistor #1 connector (back panel)		
J7-14	J1-2	Thermistor #1 output	
J7-N	J1-27	Thermistor #1 output	
J7-15	J1-1	Thermistor #0 output	
J7-P	J1-26	Thermistor #0 output	
P7-16	J1-4	Thermistor #3 output	
J7-R	J1-29	Thermistor #3 output	
J7-17	J1-3	Thermistor #2 output	
J7-S	J1-28	Thermistor #2 output	
J7-18	J1-5	Thermistor #4 output	
J7-T	J1-30	Thermistor #4 output	
J7-19	J1-6	pH output	
J7-20	J1-9	T2 output	
J7-21	J1-42	Irrigate	
J7-22	J8-E	Reset CO_2 output	
J7-C	J1-39	Reset transpired water	
J7-D	J2-2	T_1	
J7-U	J1-7	Light output	
J7-W	J1-8	T1 output	
J7-X	J8-H	Reset transpired water	
J8-3	J2-5	O_2 drive	
J8-14	J2-6	CO_2 drive	
J8-5	J2-8	Irrigate drive	
J8-6	J1-21	O_2	
J8-7	J1-20	CO_2	
J8-9	J1-18	Base	
J8-10	J1-17	Acid	
J8-13	J2-9	Acid drive	
J8-14	J2-10	Base drive	
J8-15	J1-46	Electrical conductivity	
J8-16	J1-22	Red	
J8-17	J1-23	White	Electrical conductivity output
J8-18	J1-47	Black	
J8-21	J1-48	Green	
J8-19	J1-19	Transpired water	
J8-22	J2-7	Transpired water	

Backplane

From	To	Signal Name	
J8-T	J9-1	Red	Electrical conductivity input
J8-U	J9-2	White	
J8-V	J9-3	Black	
J8-W	J9-4	Green	
J10-1	Neon bulb power switch (common)	Front panel (cable to transformer card)	
J10-16	Neon bulb power switch (common)		
J10-11	Neon bulb power switch (common)	Front panel (cable to transformer card)	
J10-2	Power switch (N.O.)		
J10-14	Power switch (N.O.)		
J10-3	Impeller test point, impeller switch (common)		
J10-4	Impeller switch (N.O.)		
J10-5	Cart enable lamp		
J10-6	Impeller pot. (high)		
J10-7	Ground post, cart enable lamp, impeller pot (low), fan pot (low)		
J10-15			
J10-8	Fan pot. (high)		
J10-9	Fan test point, fan switch (common)		
J10-10	Fan switch (N.O.)		
J10-12	Neon light		
J10-13	Neon light		
J11-1	pH test point	pH output	
J11-8	Set pot. (c.t.)		
J11-9	Amp pot. (c.w. & c.t.)		
J11-11	Set pot. (c.w.)	+6V	
J11-13	Amp pot. (c.c.w.)		
J11-14	Set pot. (c.c.w.)	-6V	

ELECTRONICS MODULE PANEL VIEWS

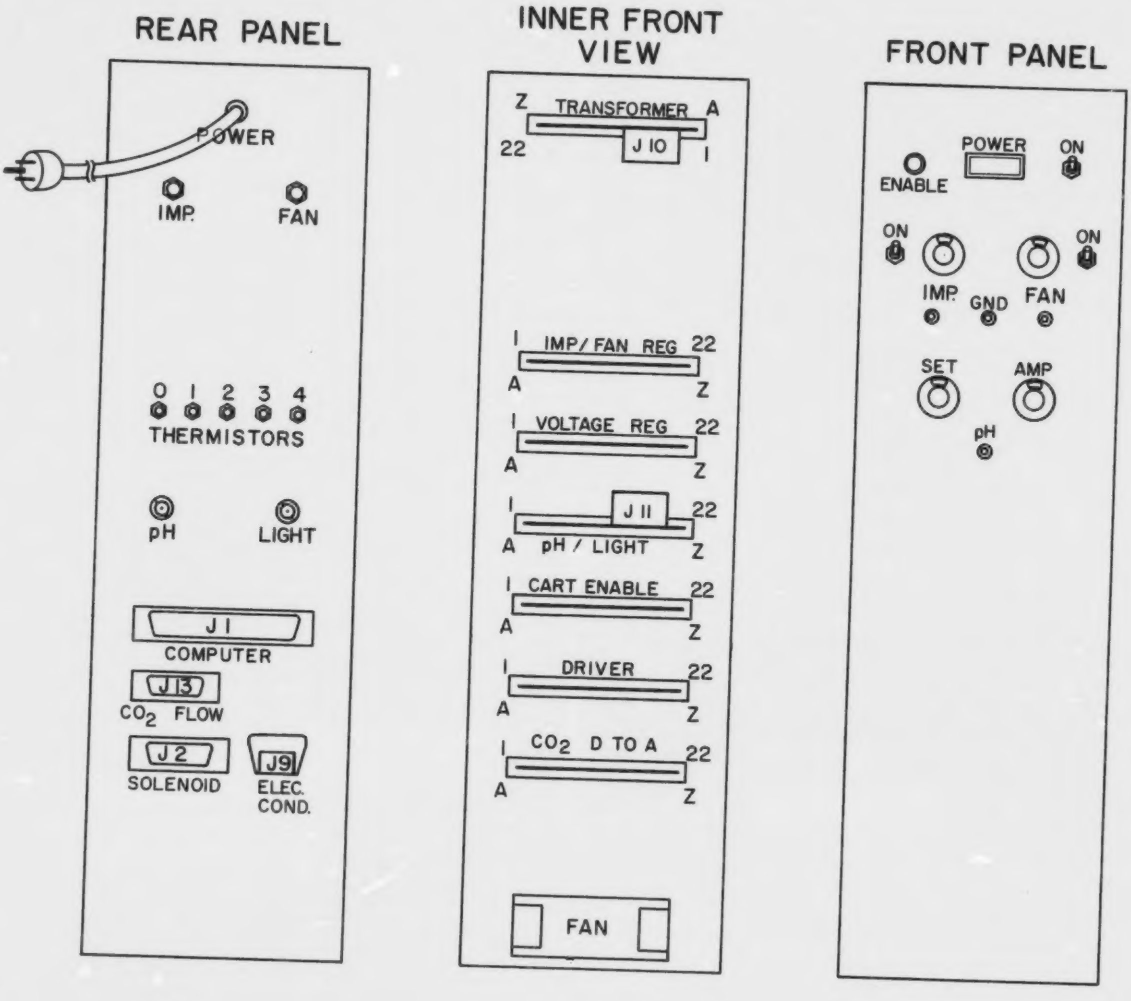

REAR PANEL
POWER
IMP.
FAN
0 1 2 3 4
THERMISTORS
pH
LIGHT
J1
COMPUTER
J13
CO2 FLOW
J2
SOLENOID
J9
ELEC. COND.
INNER FRONT VIEW
Z TRANSFORMER A
22
J 10
1
1 IMP/ FAN REG 22
A
Z
1 VOLTAGE REG 22
A
Z
1
J 11
22
A pH / LIGHT Z
1 CART ENABLE 22
A
Z
1 DRIVER 22
A
Z
1 CO2 D TO A 22
A
Z
FAN
FRONT PANEL
ENABLE
POWER
ON
ON
ON
IMP.
GND
FAN
SET
AMP
pH

Figure 4-9. Connector locations for electronics module. The locations for the card edge connectors are transformer J3, imp/fan reg J4, voltage reg J5, pH/light J6, cart enable J7, driver J8, and CO_2 D to A J12.

Appendix 4

Computer Connector

1. Therm. #0
2. Therm. #1
3. Therm. #2
4. Therm. #3
5. Therm. #4
6. pH
7. Light
8. T1
9. T2
10. LSB
11. LSB+1
12. LSB+2
13. LSB+3 CO_2 Digital
14. LSB+4 Code
15. LSB+5
16. LSB+6
17. Acid
18. Base
19. Transpired H_2O (on)
20. Not used
21. O_2
22. (E.C. red)
23. (E.C. white)
24. LSB+7 CO_2 Digital
25. MSB Code
26. Therm. #0
27. Therm. #1
28. Therm. #2
29. Therm. #3
30. Therm. #4
31. Ground
32. Ground
33. Ground
34. Ground
35. Ground
36. Ground
37. Ground
38. Not used
39. Transpired H_2O (off)
40. Cart enable
41. Not used
42. Irrigation
43. Assigned to gas circuits
44. Assigned to gas circuits
45. Assigned to gas circuits
46. Elect. cond. (on/off)
47. (E.C. black)
48. (E.C. green)
49. Load CO_2 code
50. CO_2 feedback voltage

25 Pin Connector for Cart Wiring Harness

1. 12V
2. T1 (solution level)
3. 12V
4. T2 (transpired water level)
5. O_2 drive
6. CO_2 drive
7. Transpired water drive
8. Irrigate drive
9. Acid drive
10. Base drive

14-25. Ground

Mass Flow Controller (D to A Card)

The level of carbon dioxide in a chamber is maintained by a mass flow controller, the accompanying electronics and a routine from the computer program.

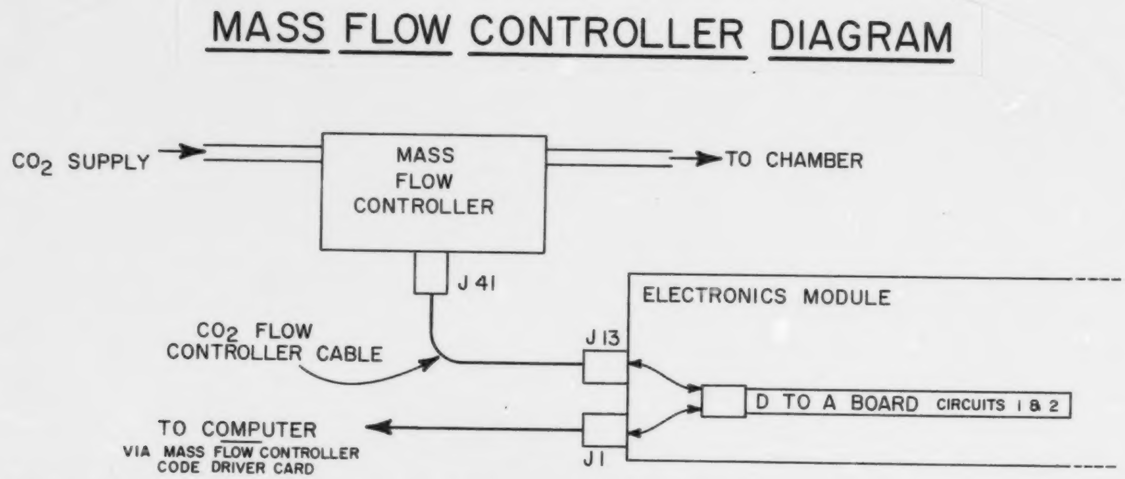

MASS FLOW CONTROLLER DIAGRAM
CO2 SUPPLY
MASS FLOW CONTROLLER
TO CHAMBER
J 41
ELECTRONICS MODULE
CO2 FLOW CONTROLLER CABLE
J 13
D TO A BOARD CIRCUITS 1 & 2
TO COMPUTER
VIA MASS FLOW CONTROLLER CODE DRIVER CARD
J 1

Figure 4-10. Mass flow controller. Wiring and plumbing of the controller unit.

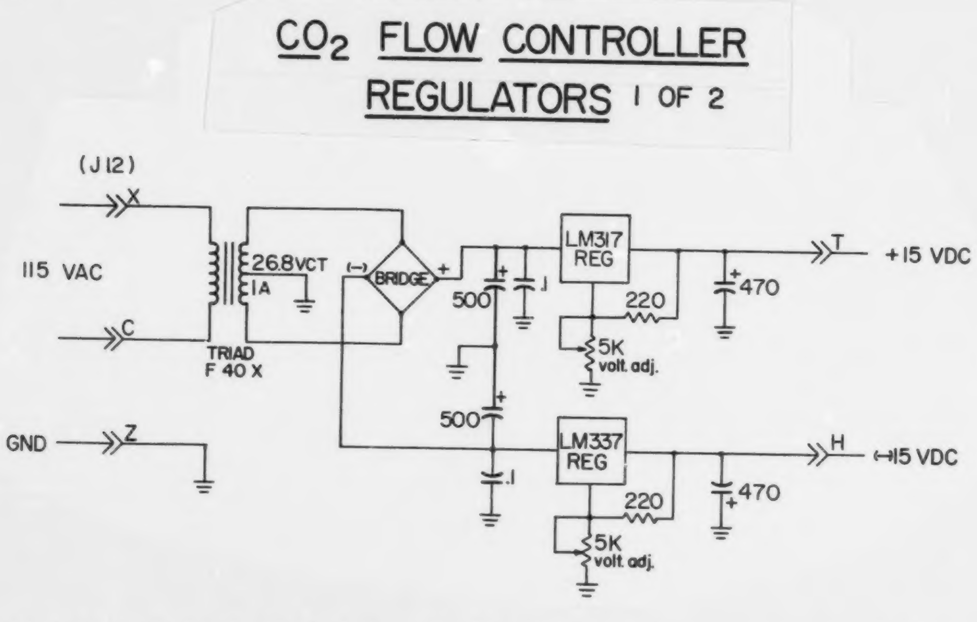

CO2 FLOW CONTROLLER REGULATORS 1 OF 2
(J12)
X
115 VAC
C
26.8VCT
1A
TRIAD
F 40 X
(–)
BRIDGE
+
500
.1
500
.1
LM317 REG
220
5K volt. adj.
470
T
+15 VDC
GND
Z
LM337 REG
220
5K volt. adj.
470
H
(–)15 VDC

Figure 4-11. D to A card (circuit I). The ±15 VDC power supply for the D to A circuitry and flow controller.

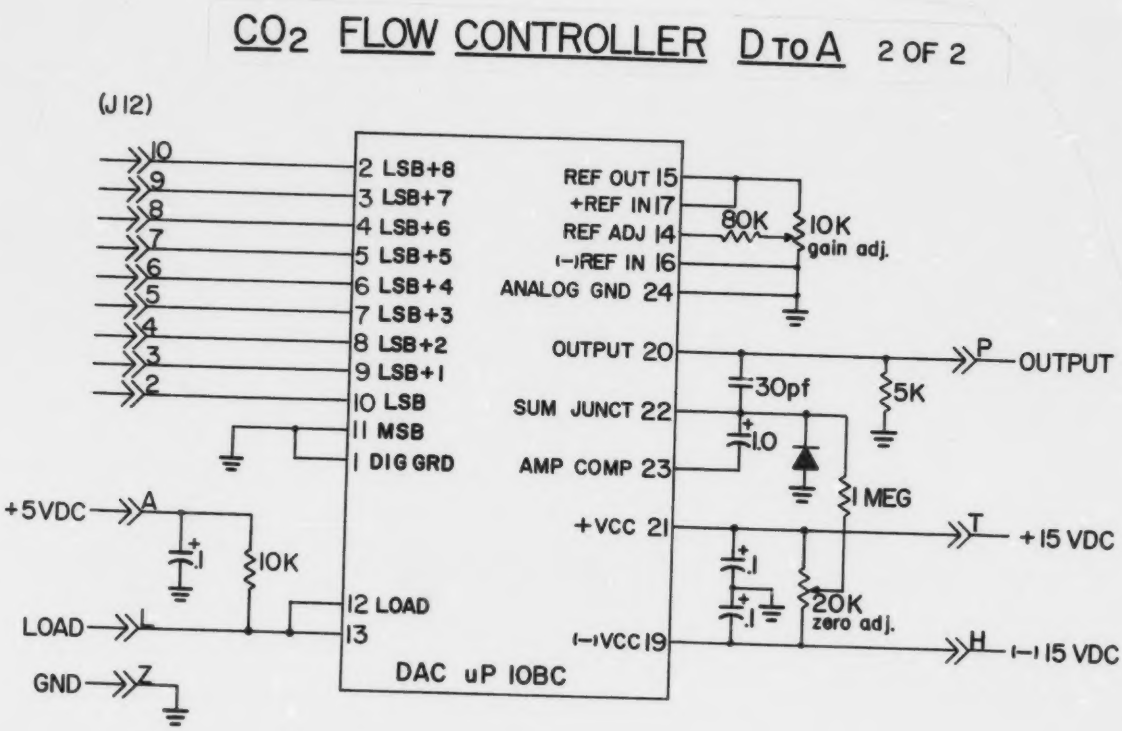

CO2 FLOW CONTROLLER D TO A 2 OF 2
(J12)
10
9
8
7
6
5
4
3
2
2 LSB+8
3 LSB+7
4 LSB+6
5 LSB+5
6 LSB+4
7 LSB+3
8 LSB+2
9 LSB+1
10 LSB
11 MSB
1 DIG GRD
12 LOAD
13
DAC uP 10BC
REF OUT 15
+REF IN 17
REF ADJ 14
(−)REF IN 16
ANALOG GND 24
OUTPUT 20
SUM JUNCT 22
AMP COMP 23
+VCC 21
(−)VCC 19
80K
10K
gain adj.
30pf
1.0
5K
1 MEG
20K
zero adj.
.1
.1
P
OUTPUT
T
+15 VDC
H
(−) 15 VDC
+5VDC
A
.1
10K
LOAD
L
GND
Z

Figure 4-12. D to A card (circuit II). This is the bottom card in the electronics module. The digital signals arrive to this integrated circuit from the Apple Computer DIO9 card via the 50-wire cord to the backplane connector J12 (pins 2-10 and L). Input J12-L comes from the cart enable card before entering the backplane to isolate this flow controller from all others on line. Output J12-P goes out through the cart enable card to also isolate this signal from other carts.

Alignments -- (1) Pins 2-10 are all tied to 0.0 VDC, then J12-L is strobed to 0.0 VDC, +5.0 VDC, and 0.0 VDC in succession. With a voltmeter between J12-P and the ground, adjust the zero adjust potentiometer until J12-P is at 0.0 VDC. (2) Pins 2-10 all are tied to +5.0 VDC, then J12-L is strobed to 0.0 VDC, +5.0 VDC, and 0.0 VDC again. Adjust the gain pot until J12-P is at 4.99 VDC. Repeat steps 1 and 2; make sure they are correct. J12-P is the output to the flow controller. As the digital codes change and J12-L is strobed by the computer, the flow controller varies the CO_2 flow accordingly. Power for the flow controller comes in on pins J12-T, H, A, and Z (ground).

CO$_2$ FLOW CONTROLLER CABLE

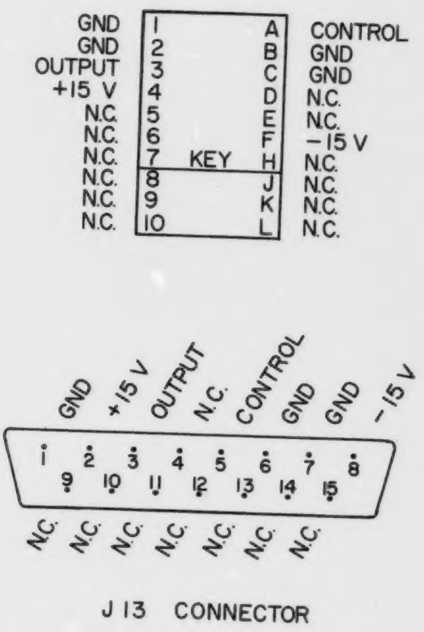

GND 1 A CONTROL
GND 2 B GND
OUTPUT 3 C GND
+15 V 4 D N.C.
N.C. 5 E N.C.
N.C. 6 F −15 V
N.C. 7 KEY H N.C.
N.C. 8 J N.C.
N.C. 9 K N.C.
N.C. 10 L N.C.
GND
+15 V
OUTPUT
N.C.
CONTROL
GND
GND
−15 V
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15
N.C.
N.C.
N.C.
N.C.
N.C.
N.C.
N.C.
J13 CONNECTOR

Figure 4-13. Mass flow controller cable (CO$_2$)

FROM		TO	SIGNAL NAME
J1-10	to	J12-2	LSB
J1-11		J12-3	LSB+1
J1-12		J12-4	LSB+2
J1-13		J12-5	LSB+3
J1-14		J12-6	LSB+4
J1-15		J12-7	LSB+5
J1-16		J12-8	LSB+6
J1-24		J12-9	LSB+7
J1-25		J12-10	MSB
J1-49		J7-2	Load CO$_2$ Code (comp.)
J1-50		J7-B	Feedback (comp.)
J12-P		J13-5	Control
J12-L		J7-4	Load CO$_2$ Code (D to A)
J12-T		J13-2	+15 VDC
J12-H		J13-8	-15 VDC
J12-Z		J13-1,6,7,J8-Z	GND
J7-Y		J13-3	Feedback (D to A)
J12-H		J13-8	-15 VDC
J12-A		J8-A	+15 VDC

CODE DRIVER CARD

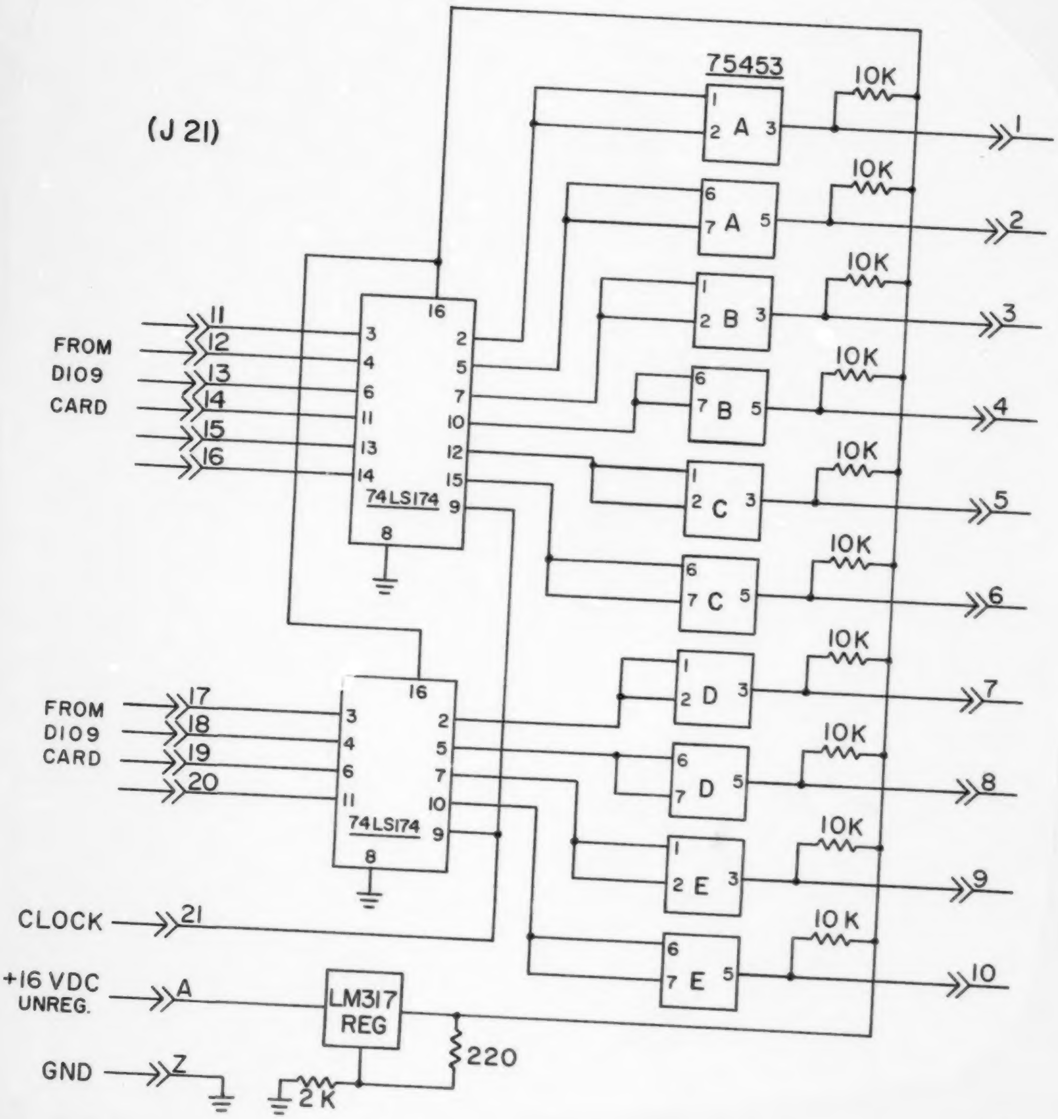

(J 21)
FROM DIO9 CARD
11
12
13
14
15
16
FROM DIO9 CARD
17
18
19
20
CLOCK
21
+16 VDC UNREG.
A
GND
Z
74LS174
74LS174
75453
A
B
C
D
E
10K
LM317 REG
220
2K
1
2
3
4
5
6
7
8
9
10

Figure 4-14. Mass flow controller code driver card. Codes, coming from the DI09 located inside the computer, are latched into the integrated circuit (75453) to eliminate loading effect. The outputs then go to a bus which goes to all the carts. The specific cart is selected with a card enable signal which allows the desired mass flow controller to be adjusted by the computer.

INTERFACE MAIN POWER

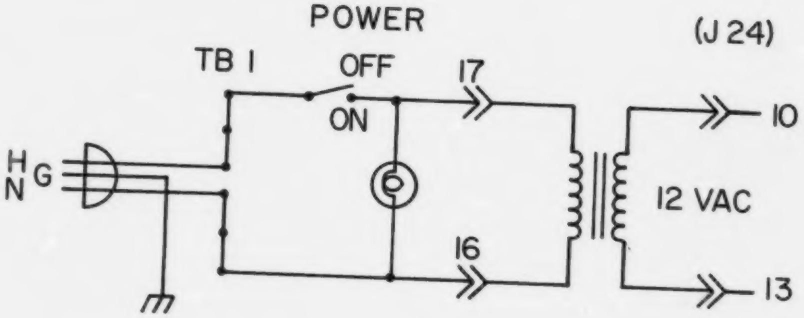

POWER
(J 24)
TB 1
OFF
17
ON
10
H
G
N
12 VAC
16
13

Figure 4-15. Interface main power (transformer card #2). 115 VAC is input to the transformer which supplies 12 VAC to power 12 VAC central cabinet power supply card #2. It also supplies 115 VAC to the ON/OFF switch and the power light on the front panel.

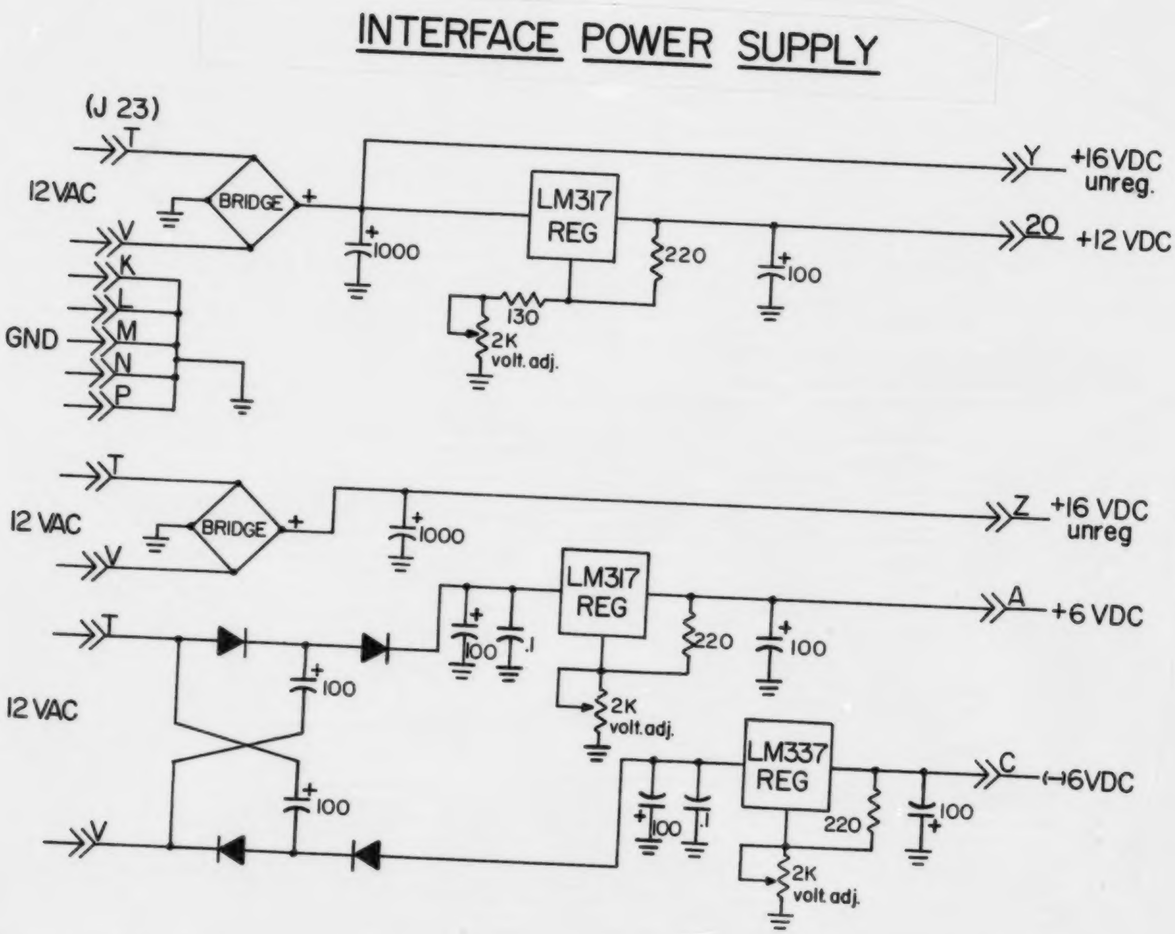

INTERFACE POWER SUPPLY
(J 23)
T
12 VAC
V
BRIDGE
+
1000
LM317
REG
220
100
130
2K
volt. adj.
Y
+16 VDC
unreg.
20
+12 VDC
K
L
GND
M
N
P
T
12 VAC
V
BRIDGE
+
1000
Z
+16 VDC
unreg
LM317
REG
100
.1
220
100
A
+6 VDC
2K
volt. adj.
T
12 VAC
100
100
V
LM337
REG
100
.1
220
100
2K
volt adj.
C
(–)6VDC

Figure 4-16. Power supply card #2. The card provides regulated +6 VDC, -6 VDC, +12 VDC, and +16 VDC unregulated to the other cards in the central electronics cabinet. The input is 12 VAC.

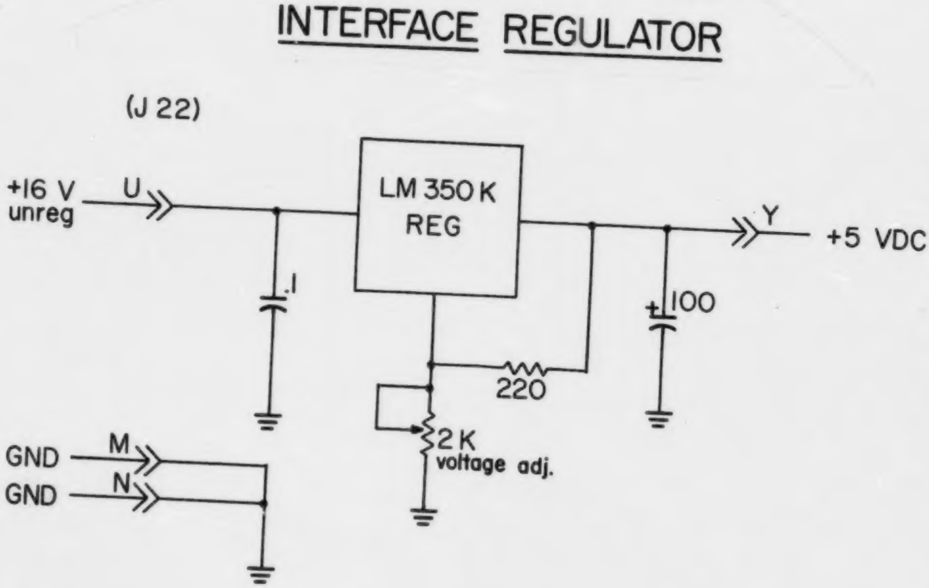

INTERFACE REGULATOR
(J 22)
+16 V
unreg
U
LM 350 K
REG
Y
+5 VDC
.1
+
100
220
2 K
voltage adj.
GND
M
GND
N

Figure 4-17. Regulator card #2. Provides regulated 5 VDC for the gas enable board and the other boards in the central electronics cabinet. The input is 16 VDC unregulated.

Cart Enable Select

Signals are isolated to prevent signals from different carts affecting each other. For example the transpired water level signal line for all carts goes to Channel 8 on the A to D converter. If this signal were not isolated by cart, the voltage would be meaningless. By disabling all the carts and measuring them individually, the signals then become valid. This is accomplished by using a series of relays on the cart enable card found in each electronics module. The cart signals come to one side of the relay contacts and the other side goes to the A to D converter channels. As long as the relay is open, no signal goes to the A to D converter.

When the computer selects a cart, it sends the DI09 the proper code. The DI09 codes are sent to the cart enable select which turns on only 1 of 18 lines corresponding to a particular cart. The integrated circuit 74LS174 latches the code and holds that cart enabled. A light on the front of the cart signals when the cart is enabled. Code "63" latches in the highest number in the integrated circuit and selects a non-existing cart, thus disabling all carts.

CART ENABLE SELECT

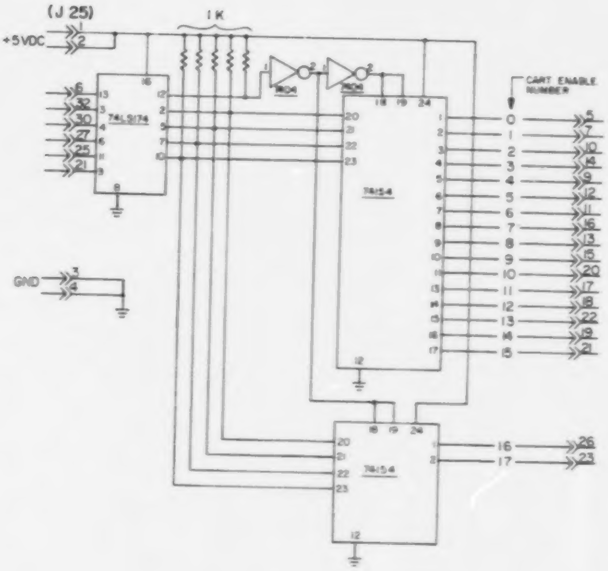


Figure 4-18. Cart enable select.

Gas Enable Select

Gas enable uses the same technique as cart enable select. The integrated circuit 7430 determines on which side of the laboratory the cart is located and drives relays which operate solenoids routing gases through appropriate plumbing.

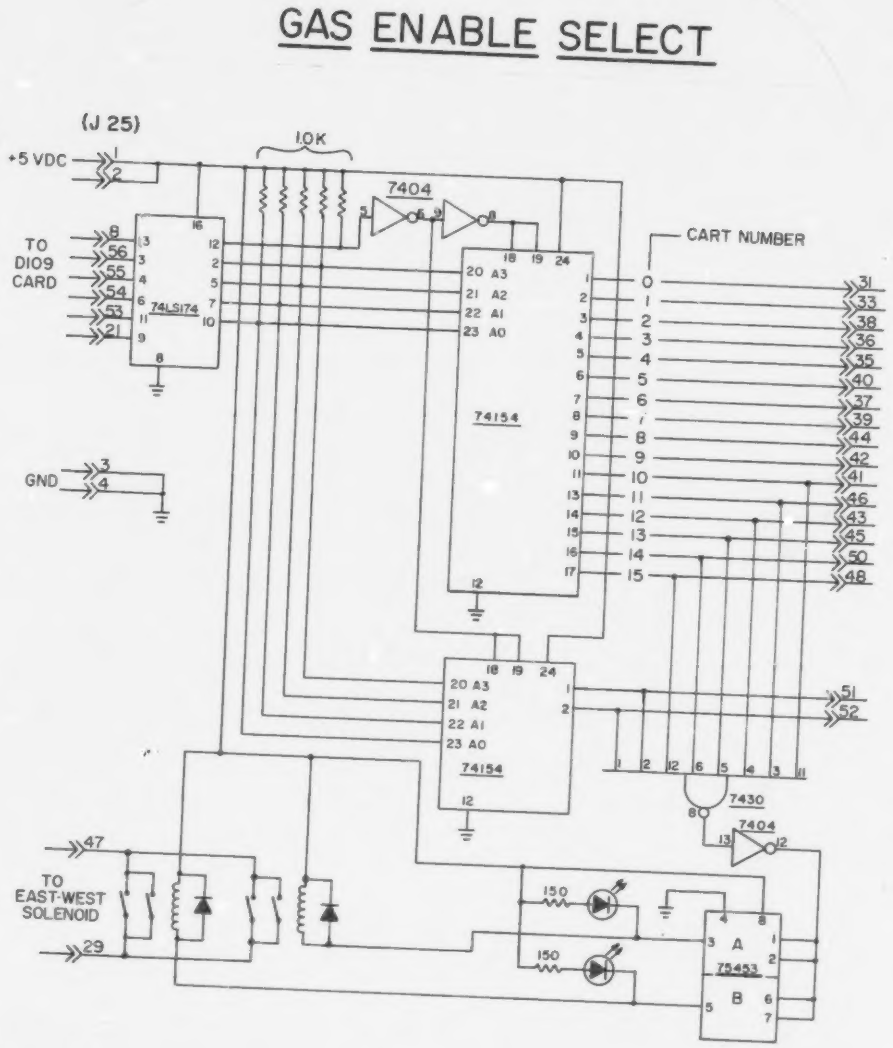

GAS ENABLE SELECT
(J 25)
+5 VDC
GND
TO DIO9 CARD
1.0K
7404
74LS174
74154
CART NUMBER
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
31
33
38
36
35
40
37
39
44
42
41
46
43
45
50
48
51
52
7430
7404
TO EAST-WEST SOLENOID
47
29
150
150
75453
A
B

Figure 4-19. Gas enable select.

GAS ENABLE SOLENOID DRIVER

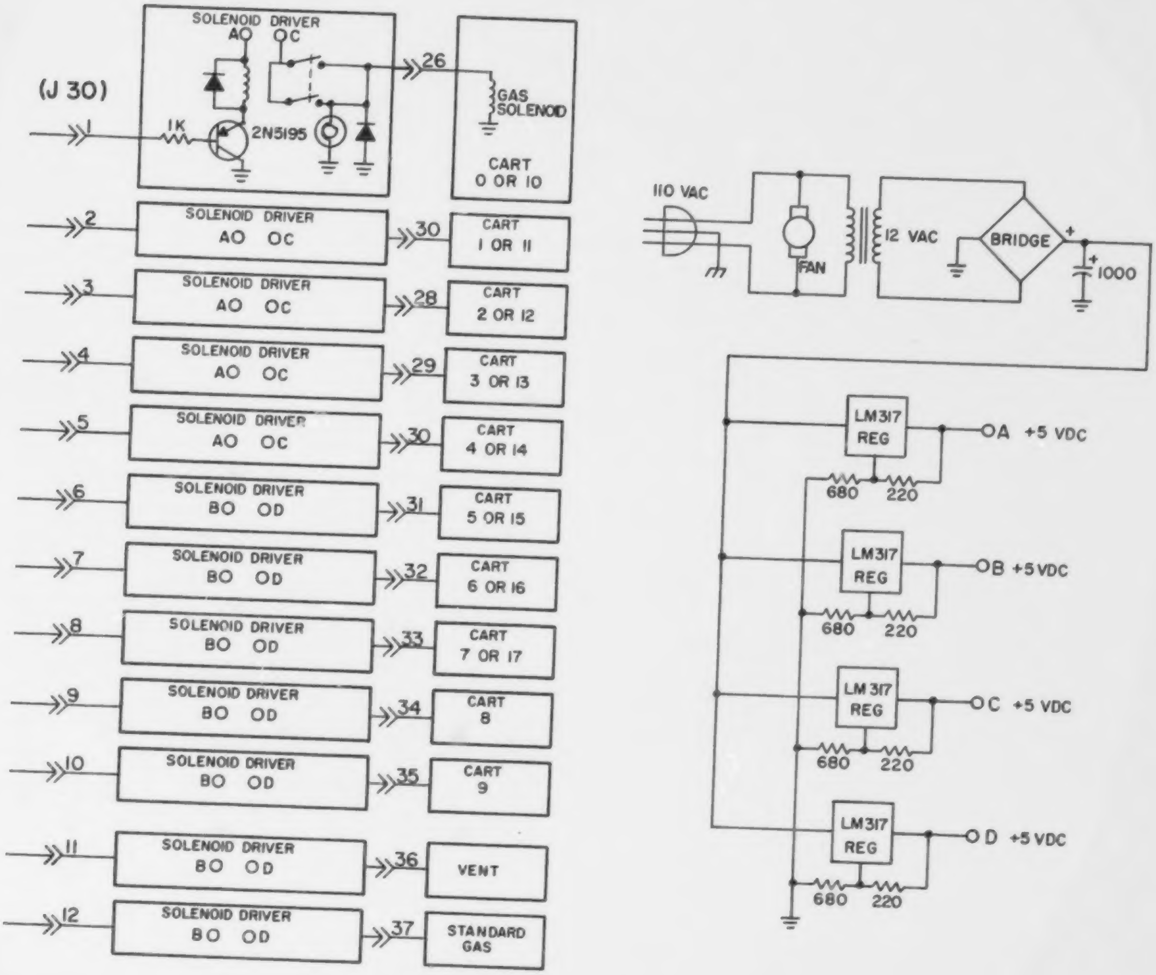

(J 30)
SOLENOID DRIVER
AO OC
1K
2N5195
26
GAS SOLENOID
CART 0 OR 10
SOLENOID DRIVER AO OC
30
CART 1 OR 11
28
CART 2 OR 12
29
CART 3 OR 13
30
CART 4 OR 14
SOLENOID DRIVER BO OD
31
CART 5 OR 15
32
CART 6 OR 16
33
CART 7 OR 17
34
CART 8
35
CART 9
36
VENT
37
STANDARD GAS
110 VAC
FAN
12 VAC
BRIDGE
1000
LM317 REG
680
220
A +5 VDC
B +5 VDC
C +5 VDC
D +5 VDC

Figure 4-20. Gas enable solenoid driver. The gas enable circuits select which cart should have gas measured. The (0.0 VDC to 5.0 VDC) signal comes to the solenoid driver which energizes a relay. The relay energizes the solenoids for that particular cart. It also turns on the appropriate indicator light on the gas enable solenoid driver box.

Dual regulators are used for each of the voltages so loads can be divided, eliminating power load problems when more than one solenoid is energized.

LITERATURE CITED

Dobkin, Robert C. (1973). Figure. "Current to Voltage Converter" from OP Amp Circuit Collection, Section 1 - Basic Circuits, page AN-31-2 IN: Linear Applications, Marvin K. Vanderkooi, ed. National Semiconductor Corporation, Santa Clara, Calif.

Lancaster, Don (1982). CMOS Cookbook. Howard W. Sams and Co., Inc. Indianapolis. Seventh edition. pp. 347-349.

APPENDIX 5
THERMISTOR CONSTRUCTION AND CALIBRATION

by

Craig Mc Farlane

Ray Shimabuku

Toxic and Hazardous Materials Branch
Corvallis Environmental Research Laboratory
200 S.W. 35th Street
Corvallis, Oregon 97333

and

John Wilson

Division of Allied Bendix Aerospace
Bendix Field Engineering Corp.
Corvallis Environmental Research Laboratory
200 S.W. 35th Street
Corvallis, Oregon 97333

CORVALLIS ENVIRONMENTAL RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CORVALLIS, OREGON 97333

DISCLAIMER

The information in this document has been funded wholly or in part by the U.S. Environmental Protection Agency. It has been subjected to the Agency's peer and administrative reviews and has been approved for publication as an EPA document. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

Appendix 5

THERMISTOR CONSTRUCTION AND CALIBRATION

CONSTRUCTION

Matched thermistors were purchased and individually packaged by ourselves in stainless steel tubes to form a temperature probe with a 1/8" phone plug for connection to the electronic package. Commercially available probes were regarded as too expensive when considering the number required. They were made of readily available materials according to the following instructions:

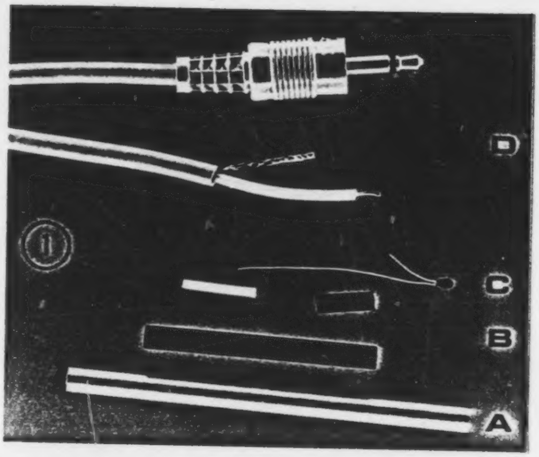
Materials:

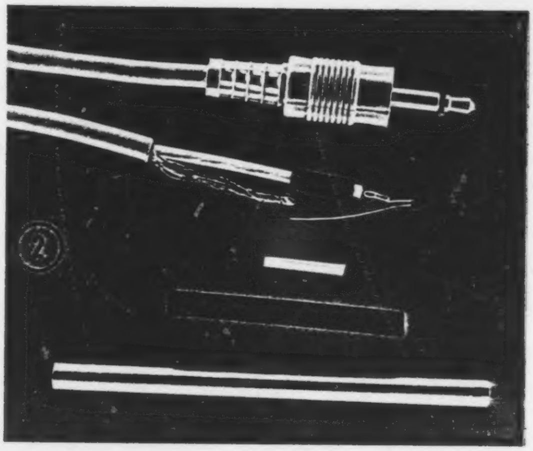
Thermistors: Uni-curve R-T molded (Newark Electronics #30F1797)
Cable: one wire shielded cable, having 1/8" mini-phone plugs on each end (Radio Shack)
Heat shrink tubing
Stainless steel tubing 3/16" OD, 0.016 wall
Thermite heat sink compound

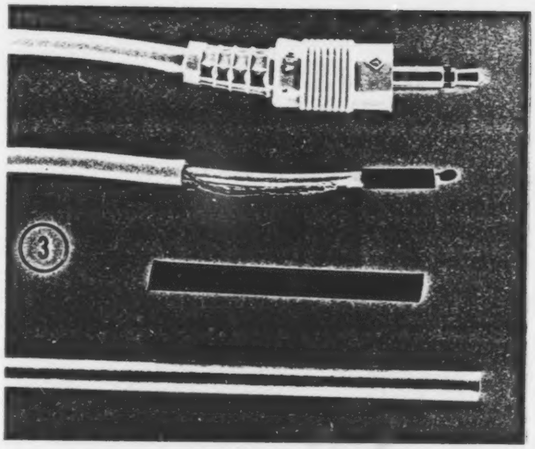
Instructions: (Figure 5-1 shows a pictorial sequence of assembly)

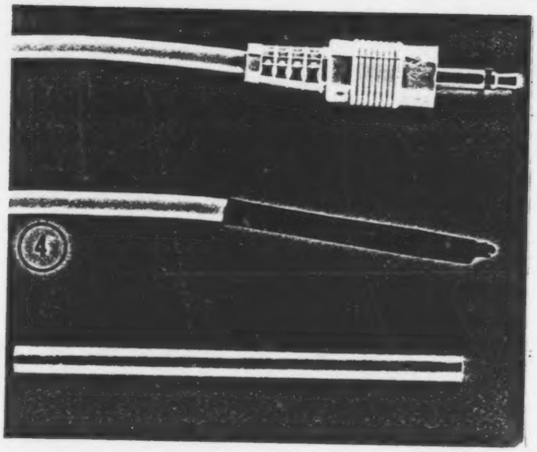
Cut the cable to desired lengths. Remove 5 cm of the outer cover and gather the braided shielding wires together. Remove 1 cm of the inner insulation. Slip on a piece of heat shrink tubing, solder the inner wire to a 1-cm long thermistor wire, and shrink the tubing over the bare wire. Solder the second wire to the wire shield, and cover with heat shrink tubing.

Cut a piece of stainless tubing to desired length (typically 7-10 cm). Seal one end by soldering to a thin piece of stainless steel. A large soldering iron, silver solder, flux and a disc of stainless steel sheet cut with a paper punch works well. File off excess to yield a neatly sealed tube. Use a syringe with a large diameter, long needle to put a small dab of Thermite into the sealed end of the tube. This acts as a heat conductor between the tube and thermistor. Slide the thermistor into the tube and seal between the cable and tube with epoxy.









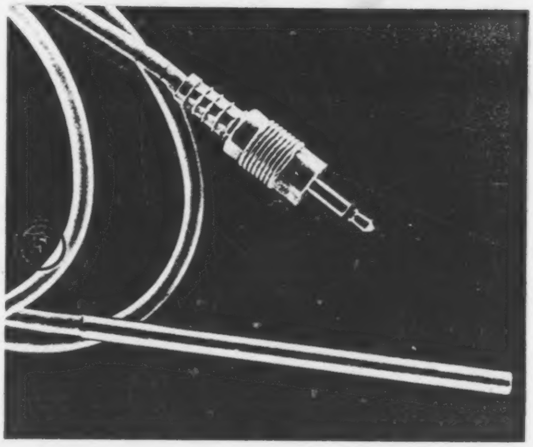


Figure 5-1. Thermistor assembly sequence.

1. All pieces of the thermistor probe ready for assembly: (A) stainless steel tube, (B) three pieces of heat shrink tubing, (C) thermistor, (D) shielding cable with 1/8" mini-phone plug.
2. Small shrink tube slipped over center wire and short thermistor lead soldered.
3. Second thermistor lead soldered and shrink tube inplace.
4. Long shrink tube in place.
5. Finished thermistor probe.

Appendix 5

CALIBRATION

Thermistors were calibrated in the following manner: Twenty thermistors were suspended by a styrofoam float in a vigorously stirred water bath. One was a lab standard which was previously calibrated against a NBS traceable quartz thermometer. Crushed ice was added to yield water at 0C. A heater was then turned on which changed water temperature at about 10C per hour. The automatic data collection system was set to record the temperature of the standard thermistors and voltage of each additional thermistor at 1 minute intervals. After allowing the temperature to increase to about 50C, the thermistors were removed and the data was analyzed. A fourth order polynomial regression was fit via least squares analysis to the data of temperature (as measured by our lab standard) vs. voltage for each thermistor (Fig. 5-2). The coefficients of the polynomial were put into a table in the CONTROL program and are applied to each thermistor when needed.

$$\hat{Y} = 1.0773 + 4.8138X + 2.0043X^2 - 0.6227X^3 + 0.1071X^4$$

$$R^2 = .9999$$

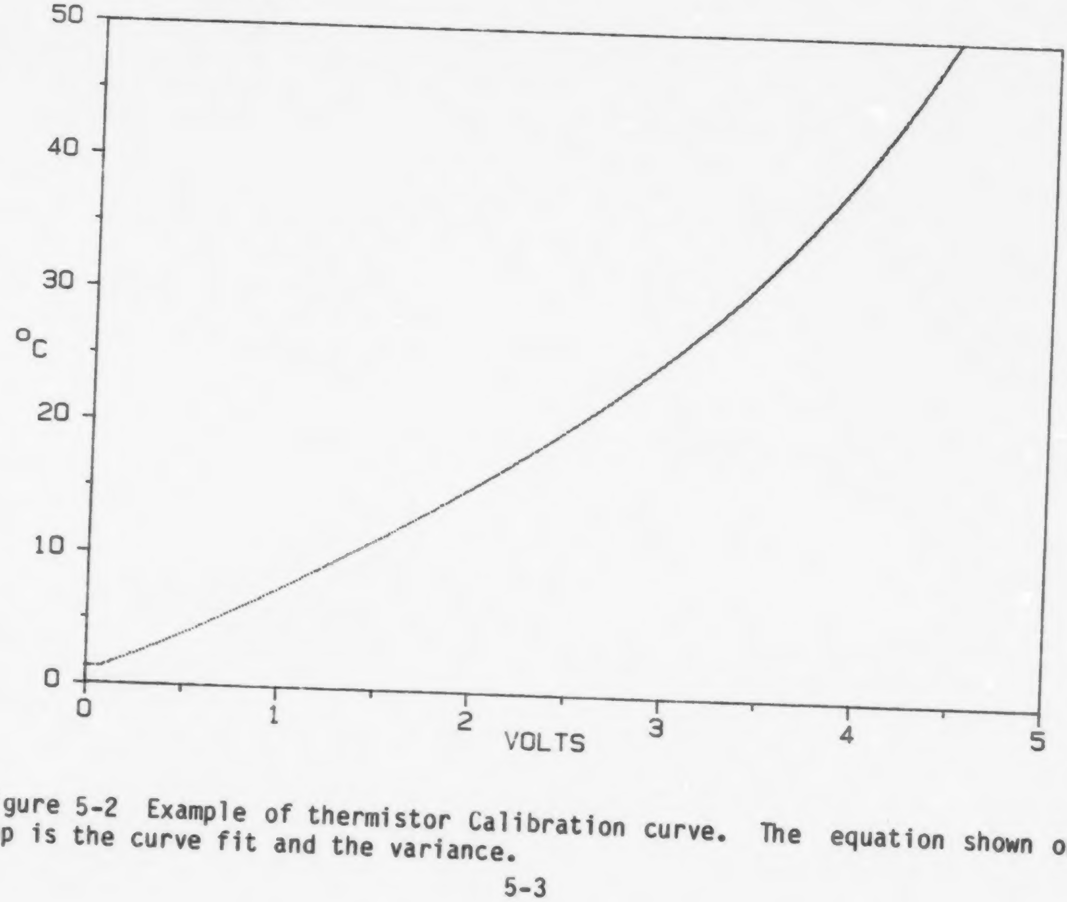


Figure 5-2 Example of thermistor Calibration curve. The equation shown on top is the curve fit and the variance.

APPENDIX 6

HYDROPONIC PLANT NURSERY CONSTRUCTION AND NUTRIENT RECIPES

by

Craig Mc Farlane

and

Kim Hayden

Toxic and Hazardous Materials Branch
Corvallis Environmental Research Laboratory
200 S.W. 35th St.
Corvallis, Oregon 97333

CORVALLIS ENVIRONMENTAL RESEARCH LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
CORVALLIS, OREGON 97333

DISCLAIMER

The information in this document has been funded wholly or in part by the U.S. Environmental Protection Agency. It has been subjected to the Agency's peer and administrative reviews and has been approved for publication as an EPA document. Mention of trade names or commercial products does not constitute endorsement or recommendation for use.

CONTENTS

Page
Figures 6-iv
Tables 6-iv
Hydroponic Nursery Construction 6-2
    Table 6-2
    Pots 6-3
    Drainage and Supply Manifolds 6-5
    Reservoir and Pump 6-6
Nutrient Recipes 6-8
    Nutrient Solution #1 6-8
    Nutrient Solution #2 6-10
Literature Cited 6-12

FIGURES

Number

Page

6-1. Hydroponic Plant Nursery 6-1
6-2. Table Top Layout 6-2
6-3. Pot Construction 6-4

TABLES

6-1. Preparation of Stock Concentrates, Nutrient #1 6-8
6-2. Nutrient Analysis, Nutrient #1 6-9
6-3. Preparation of Stock Concentrates Nutrient #2 6-10
6-4. Nutrient Analysis, Nutrient #2 6-11

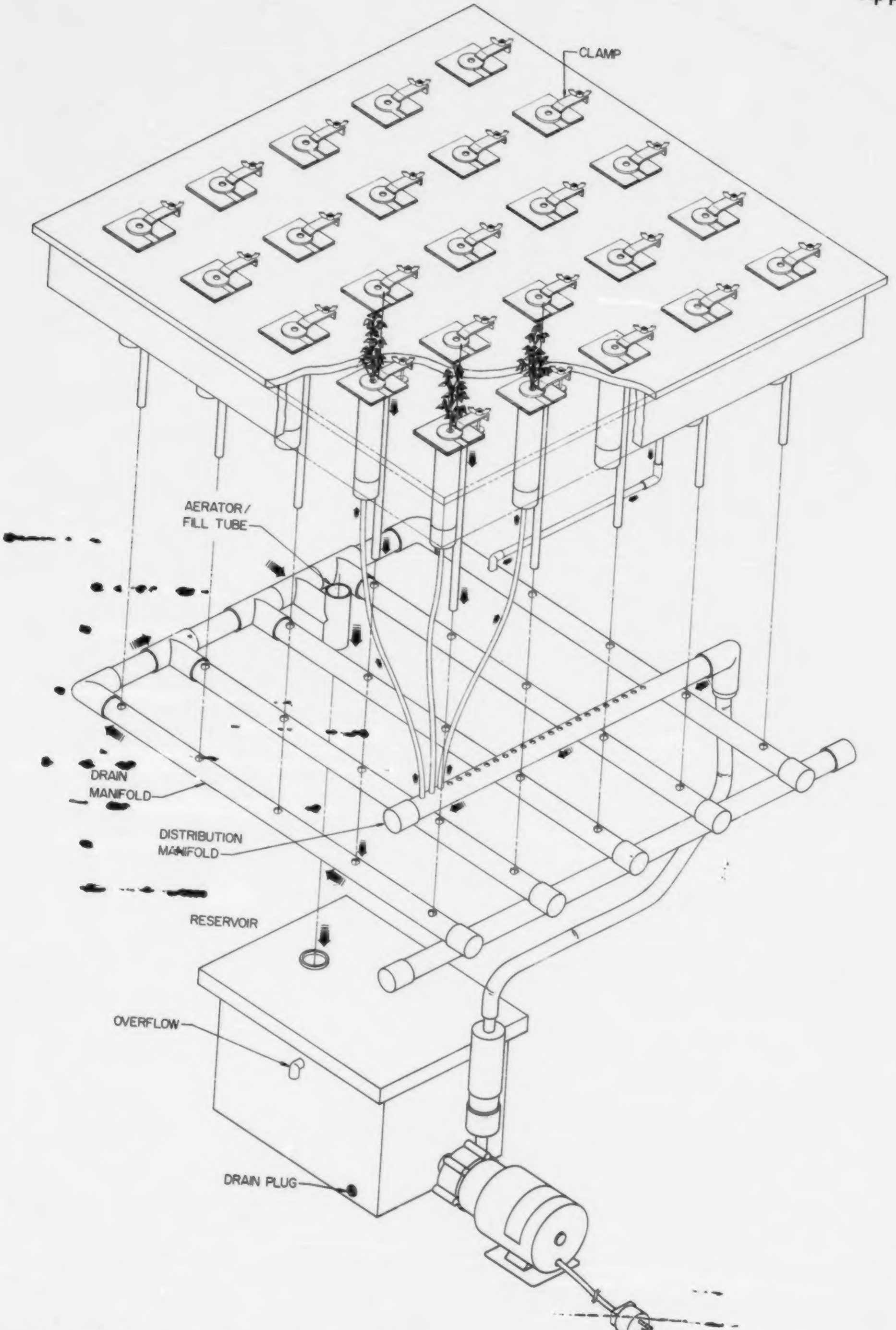

CLAMP
AERATOR/
FILL TUBE
DRAIN
MANIFOLD
DISTRIBUTION
MANIFOLD
RESERVOIR
OVERFLOW
DRAIN PLUG

Figure 6-1. Hydroponic plant nursery.

HYDROPONIC NURSERY CONSTRUCTION

Mature plants are needed for experiments on the uptake, translocation, accumulation and biodegradation of toxic organic chemicals. Plants need to be healthy, have a root geometry compatible with the exposure chambers, and be individually supported. This is accomplished in a hydroponic plant nursery (Fig. 6-1) which supplies water, mineral nutrients, and oxygen to the root. The solution flow rate is equivalent to one solution change every 1.3 minutes. Oxygen is dissolved in the solution by contact with the air during its return to the nutrient reservoir.

TABLE

Materials:

1 - 48" x 48" x 3/4" exterior plywood (table top)
4 - 6" x 45" x 3/4" exterior plywood (table support)
1 gallon white gloss epoxy paint

Instructions:

Cut 25 2-1/2" dia. holes in the plywood table top (Fig. 6-1). Additional 1-1/8" dia. holes are needed to accommodate the pot drain (Fig. 6-2). These should be at 45° to the table edges. Make sure the smaller hole just connects with the larger hole.

Make the table support by nailing the 6" plywood pieces in a square. Nail on the table top.

Paint thoroughly using white gloss epoxy paint. Make sure all exposed wood is covered.

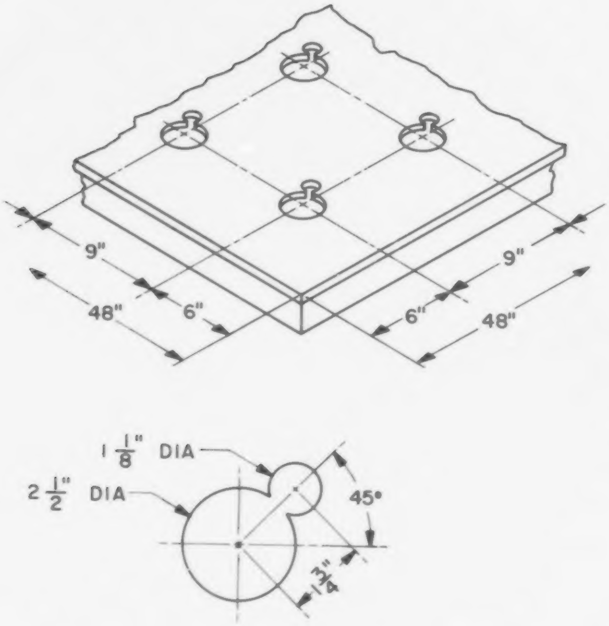

9"
9"
48"
6"
6"
48"
1 1/8" DIA
2 1/2" DIA
45°
1 3/4"

Figure 6-2. Table top layout.

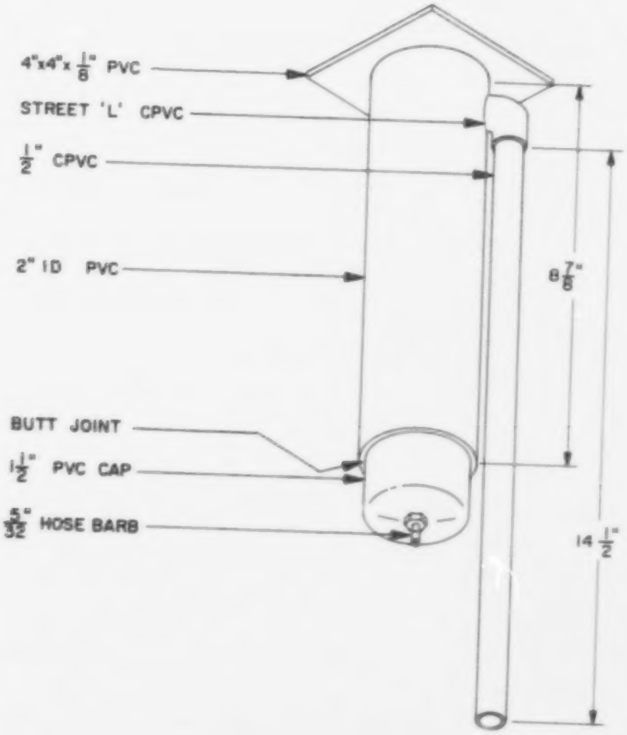

4"x4"x 1/8" PVC
STREET 'L' CPVC
1/2" CPVC
2" ID PVC
BUTT JOINT
1 1/2" PVC CAP
5/32" HOSE BARB
8 7/8"
14 1/2"

Figure 6-3. Pot construction.

POTS

Materials:

1. 23'- 2" PVC pipe (pots)
2. 40'- 1/2" CPVC pipe (overflow pipe)
3. 30 - 4"x4" gray 1/8" PVC plastic
4. 30 - 1-1/2" PVC slip caps
5. 30 - 1/2" CPVC street elbows
6. 30 - 1/8" NPT threaded hose barbs
7. CPVC/PVC primer and cement

Instructions:

To assure quality construction, always clean off burrs after drilling, cutting and grinding plastic. Use teflon anti-seize tape on all threaded fittings. Use primer and glue on all fittings unless otherwise noted. Figure (6-3) illustrates pot construction.

Cut the 2-" PVC pipe into 9" lengths. Drill a 5/8" hole (use a hole saw) that will have a center point 9/16" from a cut end.

Cut the 1/2" CPVC pipe into 15" lengths.

Cut a centered hole 2-1/2" dia. in the 4" PVC plastic squares. A jig for holding these pieces can easily be made with wood and will assure accuracy.

Drill and tap the 1-1/2" slip caps using a 5/16" drill and 1/8" NPT tap. Make sure the hole is centered in the bottom of the cap. Screw the 1/8" NPT fitting into the hole.

Glue the 1-1/2" slip caps onto the PVC pipe to form a pot. This is a butt joint and may require additional gluing.

Glue the CPVC pipe into female end of street elbow. Allow glue to dry for 15 minutes. Grind or sand male end of street elbow to 1/8".

Glue CPVC elbow with the male end into the 5/8" hole that was drilled into the 9" PVC pot. Make sure the overflow pipe is parallel with the pot. Since this is a non-standard fitting, additional gluing may be needed for a water-tight seal.

Prime and glue the base squares to the top end of the pot. The pipe should fit snuggly into the 2-1/2" hole. Make sure that one corner of the base square is in the same direction as the street elbow and overflow pipe. Additional gluing may be required.

Appendix 6

PLUMBING FOR DRAINAGE AND SUPPLY MANIFOLDS

Materials:

22' - 1-1/4" PVC pipe (125 psi)
6ea - 1/4" PVC slip caps
5ea - 1-1/4" PVC slip tees
3ea - 1-1/4" PVC slip elbows
3ea - 3/4" PVC slip elbows
35" - 1-1/4" PVC pipe (sch 40)
1ea - 1-1/4" to 3/4" PVC slip bushing
1ea - 3/4" to 1/4" NPT PVC slip bushing
25ea - 40" pieces of 1/8" ID black polyethylene tubing
25ea - 1/8" NPT threaded hose barbs
24" - 1/2" ID polyethylene tubing
1ea - 1/4" NPT ball valve
1ea - 1/4" NPT threaded hose barb

Instructions for drainage manifold:

Cut five lengths (41" long) of 1-1/4" PVC pipe. Place a 1-1/4", 90 degree elbow on one end of two of the pipes, and place the same size tee on one end of the other three pipes. Use no primer or glue on these fittings at this time. Place an end cap on the opposite ends of all five pipes.

Drill five holes on the top of each pipe 9 inches apart using a 5/8" hole saw.

Cut three 9" pieces of 1-1/4" pipe. These are used to complete three of four interconnections between the five lengths of 1-1/4" PVC pipe. The other connection is made as follows: Cut two pieces of 1-1/4" pipe (2-1/2" long) and assemble so that when fitted with three tees, there is a total of 9 inches between the middle of the first and third tees. This connection allows the manifold to drain to the reservoir. With the manifold fully assembled this connection is between the third and fourth pipe (see Fig. 6-1).

Place a 1-1/4" tee on a small piece of PVC pipe. Attach the PVC pipe to the tee between the third and fourth pipe. This 1-1/4" tee will be the aerator/fill tube opening for the nutrient reservoir.

After making sure the drainage manifold lines up with the drain pipes from the pots, glue the pipes and fittings as shown in the drawing (Fig. 6-1). Do not glue the tee going from the manifold to the reservoir.

Cut a 45" support using plastic pipe. Wire the support at the rear of the drainage system to the tables. Wire the front directly to the system. Set the five capped ends of the pipe on the support. Adjust the wire lengths so that the manifold gravity feeds back to the reservoir.

Instructions for the supply manifold:

Cut a 35" piece of 1-1/4" (sch 40) PVC pipe. Glue two 1-1/4" slip caps and one 1-1/4" slip elbow on the ends. There should be 33 inches between the fittings. Reduce the opening on the elbow end from 1-1/4" to 1/4" NPT by using two bushings (1-1/4" to 3/4" and 3/4" to 1/4" NPT).

Drill 25 holes spaced 1-1/4" apart using a 5/16" drill bit. Tap holes for 1/8" NPT threads and screw in 1/8" NPT threaded hose barbs. The holes should face the opposite direction from the elbow.

Cut 25 pieces of 1/8" black tubing into 40" lengths. Using a heat gun, force the tubing onto the threaded hose barbs.

Drill a hole using a 7/16" drill bit through both end caps and tap holes for 1/4" NPT.

Make a supply manifold drain (not shown) by inserting a 1/4" NPT ball valve on the cappen end of the manifold. Screw in a 1/4" NPT threaded hose barb to the other end of the manifold. Attach a 24" piece of 1/2" ID polyethlene tubing to the fitting. The other end of the tubing goes to the pump.

With the drainage system and pots in place, install the finished manifold on top of the drainage pipes and attach each of the 25 tubes to the threaded hose barbs on the pots using a heat gun if necessary.

RESERVOIR AND PUMP

Materials:

1ea - 11-gallon Nalgene tank and cover
1ea - March orbital magnetic drive pump (model AC-2CP-MD)
1 roll of silver or black tape (2" wide)
1ea - float valve with 1" female thread (model 5410-010)
1ea - 1/2" NPT PVC check valve
1ea - 3/4" PVC slip elbow
1ea - 3/4" gray PVC slip union
1ea - 3" long 3/8" NPT PVC nipple
1ea - 1" thread to 3/4" PVC pipe adapter
1ea - 1/4" NPT threaded hose barb
1ea - 1/8" NPT threaded hose barb elbow
2ea - 1/2" NPT to 1/4" NPT bushings
2ea - 3/4" PVC pipe (125 psi)

Instructions:

Tape the cover and horizontally tape the reservoir tank from the bottom to the top to prevent algal growth (overlap tape to prevent moisture underneath).

Three holes are needed in the tank and two in the cover. Cut the 5/8" drain hole using a hole saw on the side of the tank as close to the bottom edge as possible and fit with a rubber stopper. Drill a 5/16" hole 10" from the bottom on the same side as the drain hole and tap for a 1/8" NPT. Screw in a 1/8" NPT threaded hose barb elbow for a overflow with the opening facing up.

Cut a 5/8" hole on the side for the 3/8" NPT nipple, which will be connected to the pump. This hole will be 1-3/4" from the bottom. Tap hole with a 3/8" NPT, and insert the nipple into the reservoir and the pump.

Make sure that the 1/2-pound spring in the 1/2" NPT check valve is at its lowest resistance (by turning nut inside counter clockwise). Install a 1/2" NPT to 1/4" NPT bushing into each end. Place the check valve (spring side down) on the outlet on the pump. Insert a 1/4" NPT threaded hose barb in the opposite opening of the valve, this will be connected to the tubing from the supply manifold.

The cover needs two holes. Place the lid onto the reservoir. Using a 1-1/8" hole saw, cut the hole so the center will measure 5 inches in from the short side and 7-1/2" from the long side. Measure 2" from the opposite short side and 7-1/2" from the long side. Use a 2-1/4" hole saw for this second hole.

To install the float valve, cut a 1-1/2" piece of 3/4" PVC pipe. Glue one end into the 1" thread to 3/4" PVC pipe adapter. Insert the other end through the 1-1/8" hole of the reservoir lid and into the gray 3/4" union (threaded side down). Attach the float valve and replace the cover. The reservoir is now ready to be placed underneath the drainage system. Connect the reservoir to the nutrient solution source through the float valve using 3/4" PVC pipe and a 3/4" PVC elbow.

Insert the 1-1/4" tee (from the drainage manifold) into the hole provided on the cover of the reservoir. This is the manual fill opening and also an opening for aerating the solution.

Connect the tubing from the intake manifold to the 1/4" NPT threaded hose barb to the top end of the check valve.

Fill and test for leaks.

Table 6-1. Nutrient solution #1. Preparation of stock concentrates (Berry, 1978).

Stock concentrates #[1]	Amount per liter
Potassium nitrate (KNO_3)	50.55 g
Potassium di-hydrogen phosphate (KH_2PO_4)	27.22 g
Magnesium sulfate ($MgSO_4 . 7H_2O$)	49.30 g
Sodium chloride ($NaCl$)	5.85 g
Micronutrient concentrate	100 ml
Add water to make one liter and mix throughly to dissolve all salts.	
Micronutrient concentrate	**Grams per liter**
Boric acid (H_3BO_3)	2.85
Manganese sulfate ($MnSO_4 . H_2O$)	1.538
Zinc sulfate ($ZnSO_4 . 7H_2O$)	0.219
Copper sulfate ($CuSO_4 . 5H_2O$)	0.078
Molybdic acid (MoO_3) (85%)	0.020
Add water to make one liter and mix throughly to dissolve all salts.	
Stock concentrate #2	**Grams per liter**
Calcium nitrate ($(CaNO_3) . 4H_2O$)*	118.08
Sequestrene 330 Fe†	5.0
Make a slurry of the iron chelate in a small amount of water before adding to the calcium nitrate concentrate.	

* If commercial calcium nitrate (Norsk Hydro) is used it has a formula of $5Ca(NO_3)2\ NH_4NO_3 . 10\ H_2O$; add only 88.8 g/L to provide the right amount of nitrogen. The calcium content will be somewhat lower.

† Other iron chelates will work: Sequestrene 138 Fe at 8.3 g/L or NAFe EDTA at 4.2 g/L.

[1] Stock concentrates are proportioned at the rate of 1 ml stock conc. #1 + 1 ml stock conc. #2 to 200 ml RO purified water.

Table 6-2. Nutrient solution #1. Approximate concentration of nutrients in final solution (Berry, 1978).

	NO_3^--N	$H_2PO_4^-$P	K^+	Na^+	Ca^{++}	Mg^{++}	$SO_4^=$-S
Atomic weight	14.01	30.97	39.10	22.99	40.08	24.31	32.06
ppm	105.0	31.0	140.0	11.0	100.0	24.0	32.0
meq/L	7.5	1.0	3.5	0.5	5.0	2.0	2.0

	Cl^-	Fe^{++}	B^{+++}	Mn^{++}	Zn^{++}	Cu^{+++}	MoO_4^-Mo
Atomic weight	35.45	55.85	10.81	54.94	65.37	63.54	95.94
ppm	18.0	2.5	0.25	0.25	0.025	0.01	0.005
meq/L	0.5	0.089	0.069	0.009	0.00076	0.00031	0.00010

Table 6-3. Nutrient solution #2. Preparation of stock solutions (Downs and Thomas, 1983).

Stock concentrate #1	Grams per liter
Magnesium nitrate, $Mg\ (NO_3)_2\cdot 6H_2)$	65.0
Calcium nitrate, $Ca(NO_3)_2\cdot 4H_2)$	160.0
Sequestrene 330 Fe, 10% Fe	25.0
Stock concentrate #2	
Potassium nitrate, KNO_3	101.1
Ammonium nitrate, NH_4NO_3	40.0
Potassium phosphate, mono KH_2PO_4	12.0
Potassium phosphate, diabasic K_2HPO_4	14.0
Potassium sulfate, K_2SO_4	15.0
Sodium sulfate, Na_2SO_4	17.0
Boric acid, H_3BO_3*	0.70
Molybdic acid, $MoO_3\cdot H_2O$*	0.005
Hampene zinc, 14.5%*	0.045
Manganous chloride, $MnCl_2\cdot 4H_2O$*	0.204
Hampol copper, 90% Cu*	0.030
Sequestrene cobalt, 14% Co*	0.001
Uranine	0.750

* The compounds containing the minor elements are dissolved together before being added to the #2 stock tank.

1. Stock solutions are proportioned at the rate of 1 ml stock conc. #1 + 1 ml stock conc. #2 to 500 ml RO purified H_2O.
2. Uranine (sodium fluorescein) is added to stock #2 to give the nutrient solution a green color so that it may be distinguished from H_2O.

Table 6-4. Analysis of nutrient solution #2 (Downs and Thomas, 1983).

Element	Symbol	Source	Total ppm in the Solution
Nitrogen	N	$Mg(NO_3)_2 \cdot 6H_2O$ $Ca(NO_3)_2 \cdot 4H_2O$ NH_4NO_3 KNO_3	106.23
Phosphorus	P	KH_2PO_3, K_2HPO_4	10.41
Potassium	K	KH_2PO_3, K_2HPO_4 K_2SO_4, KNO_3	111.03
Calcium	Ca	$Ca(NO_3)_2$	54.40
Magnesium	Mg	$Mg(NO_3)_2$	12.40
Iron	Fe	Sequestrene 330 Fe	5.00
Sulfur	S	K_2SO_4, Na_2SO_4	13.19
Manganese	Mn	$MnCl_2$	0.113
Boron	B	H_3BO_3	0.24
Zinc	Zn	Hampene Zn	0.113
Copper	Cu	Hampol Cu	0.005
Cobalt	Co	Sequestrene Co	0.0003
Molybdinium	Mo	$MoO_3 \cdot 2H_2O$	0.005
Sodium	Na	Na_2SO_4	11.04

LITERATURE CITED

Berry, W. L. 1978. Nutrition, Containers, and Media. In A Growth Chamber Manual. Ed. by R. W. Langhans. Comstock Pub. Assoc. Ithaca, N.Y. 22 pg.

Downs, R. J. and J. F. Smith. 1983. Phytotron Procedural Manual. For Controlled Environment Research at the Southeastern Plant Environment Laboratory. Tech. Bul. 225 (revised). North Carolina State University. Raleigh, N.C.